HMI • SCADA • MES Software

User Manual

inductive automation
800.266.7798
www.inductiveautomation.com

Table of Contents

Part I Introduction 16

1 Welcome to Ignition 16
2 Getting Help 16
3 Licensing, Activation, and Trial Mode 16
4 Quick Start 18

Installation (Windows) 18
Installation (Linux) 19
Gateway Homepage 20
Connect to a PLC 22
Connect to a Database 22
Launch the Designer 23
Create some SQLTags 24
Create a Window 25
Launch a Client 26
Create a Transaction Group 26

Part II Overview 28

1 What is Ignition? 28
2 Architecture 28

Architecture Overview 28
System Concepts 29
Ignition Gateway 29
Ignition Designer 29
Ignition Vision Clients 30
Mobile Clients 30
Database Access 31
OPC-UA 31
SQLTags 32
Architecture Diagrams 33
Standard Architecture 33
OPC-UA Architecture 34
Clustered Architecture 35
Remote Datalogging Architecture 36
Wide-area SCADA Architecture 37
Panel Edition Architecture 38
Advanced Architecture Topics 38
Clustering 38
Vision Panel Edition 39
Remote Logging 40
Distributed SQLTags 40
Client Retargeting 41

3 Modules 41

Overview 41
OPC-UA Module 42
SQL Bridge Module 42

© 2011 Inductive Automation

Vision Module ... 43
Reporting Module ... 43
Mobile Module ... 43
OPC-COM Module ... 44
Other Modules ... 44
4 Basic Usage ... 44
Gateway Navigation ... 44
Gateway Control Utility ... 45
Web Launching ... 46
Launching Clients ... 47
Launching the Designer ... 48
Part III Gateway Configuration 50
1 Gateway Configuration Overview ... 50
2 Logging into the configuration page ... 50
3 Basics ... 50
Basic Gateway Settings ... 50
Gateway Homepage Customization ... 52
Setting the Port ... 52
Resetting the trial period ... 52
Activation ... 52
Online Activation ... 52
Offline Activation ... 53
Unactivation ... 53
Updating the License ... 53
Gateway Console ... 53
4 Projects ... 54
What is a Project? ... 54
Project Management ... 54
Project Versioning ... 55
Importing and Exporting Projects ... 56
5 Modules ... 56
Module Management ... 56
6 Databases ... 57
Databases Overview ... 57
Supported Databases ... 58
Database Connections ... 59
Creating and Editing Connections ... 59
Monitoring Connection Status ... 59
Connecting to Microsoft SQL Server ... 59
Connecting to MySQL ... 61
Database Drivers ... 61
What is JDBC? ... 61
Can I connect using ODBC? ... 62
Adding a JDBC driver ... 62
Database Translators ... 63
7 Store and Forward ... 64
Store and Forward Overview ... 64
Engine Configuration ... 64
Store and Forward for Reliability ... 66
Store and Forward for high-speed buffering ... 67

© 2011 Inductive Automation

Engine Status Monitoring ... **67**
Data Quarantining ... **68**
8 OPC ... **68**
What is OPC? ... **68**
OPC Connections ... **69**
Connecting to OPC-UA ... 69
Connecting to OPC Classic (COM) ... 70
OPC Quick Client ... **71**
Ignition OPC-UA Server ... **71**
OPC-UA Server Settings ... 71
Adding a New Device ... 71
Verifying Device Connectivity ... 72
Drivers ... 72
Allen Bradley Drivers ... 72
ControlLogix 5500 ... 72
MicroLogix 1100/1400 ... 72
PLC-5 ... 73
SLC 505 ... 74
Simulator Drivers ... 75
Generic Simulator ... 75
Allen Bradley SLC Simulator ... 76
Modbus Drivers ... 76
Modbus Ethernet ... 76
Overview ... 76
Device Configuration ... 77
Addressing ... 77
UDP and TCP Drivers ... 84
UDP and TCP ... 84
Siemens Drivers ... 86
Overview ... 86
Addressing ... 86
9 SQLTags ... **87**
SQLTags Configuration Overview ... **87**
Configuring Realtime SQLTags ... **89**
SQLTags Realtime Provider Types ... **89**
Internal Provider ... 89
Database Provider ... 89
Database Driving Provider ... 89
How SQLTags Historian Works ... **90**
Configuring SQLTags Historian ... **91**
10 Security ... **92**
Security Overview ... **92**
Authentication Profile Types ... **92**
Internal Authentication Profile ... 92
Database Authentication Profile ... 93
Active Directory Authentication Profile ... 93
AD/Internal Authentication Profile ... 93
AD/Database Authentication Profile ... 93
Managing Users, Passwords, and Roles ... **93**
Enabling SSL Encryption ... **94**
11 Alerting ... **95**
Alerting Overview ... **95**
Alert Notification ... **95**

© 2011 Inductive Automation

Alert Storage ... 96
12 Redundancy ... 96
What is Redundancy? ... 96
How Redundancy Works ... 97
Setting up Redundancy ... 98
Redundancy Settings ... 99
Database Considerations ... 101
Troubleshooting Redundancy ... 102
Part IV Project Design 105
1 Designer Introduction ... 105
2 Using the Designer ... 105
Logging into the Designer ... 105
Creating or Opening a Project ... 105
Designer UI Overview ... 105
Using the Docking System ... 106
Communication Modes ... 106
Designer Tools ... 107
Output Console ... 107
Diagnostics Window ... 107
Find / Replace ... 108
Image Manager ... 108
Query Browser ... 109
3 SQLTags ... 109
What is a SQLTag? ... 109
Types of SQLTags ... 110
Creating SQLTags ... 110
Tag Properties ... 111
General Properties ... 111
Numeric Properties ... 112
Metadata Properties ... 112
Permission Properties ... 113
History Properties ... 113
Alerting Properties ... 114
Expression/SQL Properties ... 117
Scan Classes ... 118
Tag Paths ... 119
Data Quality ... 119
Importing/Exporting using CSV ... 120
4 Project Properties ... 120
Project General Properties ... 120
Designer General Properties ... 121
Designer Window Editing Properties ... 122
Client General Properties ... 122
Client Launching Properties ... 123
Client Login Properties ... 123
Client Polling Properties ... 124
Client User Interface Properties ... 124
5 Project Scripting Configuration ... 124
Script Modules ... 124
Event Scripts ... 125
Overview ... 125

© 2011 Inductive Automation

Startup and Shutdown Scripts ... 125
Shutdown Intercept Script ... 125
Keystroke Scripts ... 126
Timer Scripts ... 126
Tag Change Scripts ... 126
Menu Bar Scripts ... 127
6 Transaction Groups ... 127
Introduction ... 127
Anatomy of a Group ... 128
Action Settings ... 128
Trigger and Handshake Settings ... 128
Advanced Settings ... 129
Items Types ... 130
Overview ... 130
OPC Item ... 131
Expression Item ... 132
SQLTag Reference ... 133
Execution Cycle ... 134
Types Of Groups ... 135
Standard Group ... 135
Block Group ... 136
Historical Group ... 137
Stored Procedure Group ... 138
7 Windows & Components ... 138
Introduction ... 138
Windows ... 139
Windows Overview ... 139
Anatomy of a Window ... 140
Typical Window Types ... 140
Window Properties ... 142
Window Security ... 144
Typical Navigation Strategy ... 144
Swapping vs Opening ... 144
Open Windows and Performance ... 145
Parameterized Windows ... 145
Components ... 146
Introduction ... 146
Creating Components ... 147
Component Palette ... 147
Custom Palettes ... 147
SQLTags Drag-n-Drop ... 147
Selecting Components ... 148
Manipulating Components ... 148
Keyboard Shortcuts ... 149
Properties ... 150
The Property Editor ... 150
Data Types ... 150
Component Customizers ... 151
Dynamic Properties ... 152
Component Styles ... 152
Quality Overlays ... 153
Touchscreen Support ... 154
Component Layout ... 155

© 2011 Inductive Automation

Property Binding **157**
Overview 157
Polling Options 159
Bidirectional Bindings 159
Indirect Bindings 159
Binding Types 160
Tag Binding 160
Indirect Tag Binding 161
SQLTags Historian Binding 161
Property Binding 162
Expression Binding 162
DB Browse Binding 162
SQL Query Binding 163
Cell Update Binding 164
Function Binding 164
Event Handlers **164**
Overview 164
The 'event' object 165
Event Types 166
Script Builders 172
Security **173**
Role-based access 173
Tag Security 173
Component Security 174
Securing event handlers 174

Part V Scripting 176

1 About Scripting 176

2 Python 176

About Python **176**
Python Tutorial **177**
Basic Syntax 177
Control Flow 179
String Formatting 180
Functions 181
Scope and Import 183
Sequences and Dictionaries 184
Exception Handling 186
Learn More 186
Python in Ignition **187**
Working with Different Datatypes 187
Component Event Handlers 191
Working with Components 191
Global Script Modules 193
Gateway vs Client Scripts 193
Timer, Keystroke, and Tag Change Scripts 193
Python Standard Library 193
Accessing Java 194

3 Expressions 195

Overview **195**
Syntax **196**

© 2011 Inductive Automation

Part VI Deployment 200

1 Licensing and Activation ... 200
2 Making Backups ... 200
3 Restoring from a Backup ... 200
4 Transferring Servers ... 200
5 Gateway Homepage Customization ... 201
6 Gateway Web Security ... 201
7 Gateway Monitoring ... 201
8 Installing a Genuine SSL Certificate ... 203

Part VII Appendix A. Components 206

1 Input ... 206
- Text Field ... 206
- Numeric Text Field ... 209
- Spinner ... 212
- Formatted Text Field ... 215
- Password Field ... 218
- Text Area ... 221
- Dropdown List ... 223
- Slider ... 228

2 Buttons ... 231
- Button ... 231
- 2 State Toggle ... 234
- Multi-State Button ... 239
- One-Shot Button ... 242
- Momentary Button ... 246
- Toggle Button ... 250
- Check Box ... 253
- Radio Button ... 255
- Tab Strip ... 258

3 Display ... 261
- Label ... 261
- Numeric Label ... 264
- Multi-State Indicator ... 268
- LED Display ... 271
- Image ... 273
- Progress Bar ... 276
- Cylindrical Tank ... 279
- Level Indicator ... 281
- Linear Scale ... 284
- Barcode ... 287
- Meter ... 290
- Compass ... 294
- Thermometer ... 298
- Document Viewer ... 301
- IP Camera Viewer ... 303

4 Tables ... 306
- Table ... 306

© 2011 Inductive Automation

List ... 314
Alert Summary Table ... 317
Tree View ... 326
Comments Panel ... 330
5 Charts ... 335
Easy Chart ... 335
Chart ... 345
Bar Chart ... 349
Status Chart ... 354
Pie Chart ... 358
Box and Whisker Chart ... 362
Gantt Chart ... 364
6 Calendar ... 367
Calendar ... 367
Popup Calendar ... 370
Date Range ... 373
Day View ... 377
Week View ... 381
Month View ... 385
7 Shapes ... 388
Circle ... 388
Rectangle ... 391
Polygon ... 394
Line ... 397
Pipe Segment ... 400
Pipe Joint ... 402
8 Misc ... 404
Container ... 404
Paintable Canvas ... 407
Sound Player ... 409
Timer ... 411
Signal Generator ... 412
9 Reporting ... 413
Report Viewer ... 413
Row Selector ... 415
Column Selector ... 418
File Explorer ... 420
PDF Viewer ... 422
Part VIII Appendix B. Expression Functions 426
1 Aggregates ... 426
groupConcat ... 426
max ... 426
maxDate ... 427
mean ... 427
median ... 427
min ... 428
minDate ... 428
stdDev ... 429
sum ... 429
2 Colors ... 429

© 2011 Inductive Automation

brighter ... 429
color ... 430
darker ... 430
gradient ... 430
3 Date and Time ... 430
dateArithmetic ... 430
dateDiff ... 431
dateExtract ... 431
dateFormat ... 432
now ... 432
timeBetween ... 432
4 Logic ... 433
binEnc ... 433
binEnum ... 433
coalesce ... 433
getBit ... 434
if ... 434
isNull ... 434
lookup ... 434
switch ... 435
try ... 436
5 Math ... 436
abs ... 436
acos ... 436
asin ... 436
atan ... 436
ceil ... 437
cos ... 437
exp ... 437
floor ... 437
log ... 437
round ... 438
sin ... 438
sqrt ... 438
tan ... 438
todegrees ... 438
toradians ... 438
6 Strings ... 439
concat ... 439
escapeSQL ... 439
escapeXML ... 439
indexOf ... 439
lastIndexOf ... 440
left ... 440
len ... 440
lower ... 441
numberFormat ... 441
repeat ... 441
replace ... 442
right ... 442
split ... 442
stringFormat ... 443
substring ... 443

© 2011 Inductive Automation

trim ... 443
upper ... 443
7 Type Casting ... 444
toBoolean ... 444
toBorder ... 444
toColor ... 445
toDataSet ... 449
toDate ... 449
toDouble ... 449
toFloat ... 450
toFont ... 450
toInt ... 450
toInteger ... 451
toLong ... 451
toStr ... 451
toString ... 451
8 Advanced ... 451
forceQuality ... 451
runScript ... 452
tag ... 452
Part IX Appendix C. Scripting Functions 455
1 About ... 455
2 system.alert ... 455
system.alert.acknowledgeAlert ... 455
system.alert.queryAlertHistory ... 456
system.alert.queryAlertStatus ... 458
3 system.dataset ... 459
system.dataset.addRow ... 459
system.dataset.dataSetToExcel ... 460
system.dataset.dataSetToHTML ... 460
system.dataset.deleteRow ... 461
system.dataset.exportCSV ... 462
system.dataset.exportExcel ... 462
system.dataset.exportHTML ... 463
system.dataset.fromCSV ... 463
system.dataset.setValue ... 464
system.dataset.toCSV ... 465
system.dataset.toDataSet ... 465
system.dataset.toPyDataSet ... 466
system.dataset.updateRow ... 467
4 system.db ... 467
system.db.beginTransaction ... 467
system.db.closeTransaction ... 468
system.db.commitTransaction ... 469
system.db.createSProcCall ... 469
system.db.dateFormat ... 471
system.db.execSProcCall ... 472
system.db.getConnectionInfo ... 472
system.db.getConnections ... 473
system.db.refresh ... 473
system.db.rollbackTransaction ... 474

© 2011 Inductive Automation

system.db.runPrepQuery ... 474
system.db.runPrepUpdate ... 475
system.db.runQuery ... 476
system.db.runScalarQuery ... 478
system.db.runUpdateQuery ... 479
5 system.file ... 480
system.file.fileExists ... 480
system.file.getTempFile ... 481
system.file.openFile ... 481
system.file.readFileAsBytes ... 482
system.file.readFileAsString ... 482
system.file.saveFile ... 483
system.file.writeFile ... 483
6 system.gui ... 485
system.gui.chooseColor ... 485
system.gui.color ... 485
system.gui.confirm ... 486
system.gui.convertPointToScreen ... 486
system.gui.createPopupMenu ... 487
system.gui.errorBox ... 489
system.gui.getOpenedWindowNames ... 489
system.gui.getOpenedWindows ... 490
system.gui.getParentWindow ... 490
system.gui.getSibling ... 491
system.gui.getWindow ... 491
system.gui.getWindowNames ... 492
system.gui.inputBox ... 492
system.gui.isTouchscreenModeEnabled ... 493
system.gui.messageBox ... 493
system.gui.moveComponent ... 494
system.gui.passwordBox ... 495
system.gui.reshapeComponent ... 495
system.gui.resizeComponent ... 496
system.gui.setTouchscreenModeEnabled ... 496
system.gui.showNumericKeypad ... 497
system.gui.showTouchscreenKeyboard ... 498
system.gui.warningBox ... 498
7 system.nav ... 499
system.nav.centerWindow ... 499
system.nav.closeParentWindow ... 500
system.nav.closeWindow ... 500
system.nav.getCurrentWindow ... 501
system.nav.goBack ... 501
system.nav.goForward ... 502
system.nav.goHome ... 502
system.nav.openWindow ... 503
system.nav.openWindowInstance ... 503
system.nav.swapTo ... 504
system.nav.swapWindow ... 505
8 system.net ... 506
system.net.getExternalIpAddress ... 506
system.net.getHostName ... 507
system.net.getIpAddress ... 507

© 2011 Inductive Automation

system.net.httpGet ... 508
system.net.httpPost ... 509
system.net.openURL ... 510
system.net.sendEmail ... 510
9 system.opc ... 512
system.opc.getServerState ... 512
system.opc.readValue ... 512
system.opc.readValues ... 513
system.opc.writeValue ... 513
system.opc.writeValues ... 513
10 system.print ... 514
system.print.createImage ... 514
system.print.createPrintJob ... 514
system.print.printToImage ... 515
11 system.security ... 516
system.security.getRoles ... 516
system.security.getUsername ... 516
system.security.isScreenLocked ... 517
system.security.lockScreen ... 517
system.security.logout ... 518
system.security.switchUser ... 518
system.security.unlockScreen ... 519
12 system.tag ... 519
system.tag.getTagValue ... 519
system.tag.getTagValues ... 520
system.tag.isOverlaysEnabled ... 520
system.tag.queryTagHistory ... 520
system.tag.setOverlaysEnabled ... 521
system.tag.writeToTag ... 522
system.tag.writeToTagSynchronous ... 522
system.tag.writeToTags ... 523
13 system.util ... 523
system.util.beep ... 523
system.util.execute ... 523
system.util.exit ... 524
system.util.getClientId ... 524
system.util.getConnectTimeout ... 525
system.util.getConnectionMode ... 525
system.util.getEdition ... 525
system.util.getGatewayAddress ... 526
system.util.getInactivitySeconds ... 526
system.util.getProjectName ... 527
system.util.getProperty ... 527
system.util.getReadTimeout ... 528
system.util.getSessionInfo ... 528
system.util.getSystemFlags ... 529
system.util.invokeAsynchronous ... 529
system.util.invokeLater ... 530
system.util.playSoundClip ... 531
system.util.queryAuditLog ... 532
system.util.retarget ... 533
system.util.setConnectTimeout ... 534

© 2011 Inductive Automation

system.util.setConnectionMode 534
system.util.setReadTimeout 535

Index 536

© 2011 Inductive Automation

Introduction

Part I

1 Introduction

1.1 Welcome to Ignition

Welcome to *Ignition by Inductive Automation*, the next generation of accessible, scalable, and data-centric HMI/SCADA/MES software. Ignition was designed from the ground up to be approachable and easy to get started with, but highly flexible and capable of scaling up to the largest projects.

This guide aims to introduce you to Ignition and its architecture, get you started quickly, and then provide all of the reference resources you should need as you become more proficient with the system.

We recommend proceeding through this manual roughly in the order that it's laid out. In particular, we recommend starting with the following topics:

What is Ignition?
Architecture Overview
Quick Start

But how do I...?

Of course, it would be impossible for us to anticipate every question or goal a reader might have, and therefore we strive to be as approachable and helpful as possible. The Getting Help section outlines a variety of ways that you can find answers to any questions that might not be answered here. Additionally, we encourage users to contact us with feedback about the product or this manual. Our goal is to always provide powerful and straight-forward software that solves problems, and to this end we rely heavily on the feedback of our users and integrator partners.

1.2 Getting Help

If you get stuck designing a system in Ignition, don't worry! There are lots of ways to get help.

Online Forum

One of the most effective ways to get help is our active user forum. The forum is always available, and is actively patrolled by Inductive Automation staff and many knowledgeable users. Chances are you will find your question already answered in an existing post, but if not you can be assured that yours will receive a quick reply. The forum can be found under the *Support* section of the Inductive Automation website.

Phone Support

You can reach us during business hours 8am-5pm PST at 1-800-266-7798. Support charges may apply. 24-hour support is also available, at an additional fee.

E-Mail Support

E-mail support is available at support@inductiveautomation.com

1.3 Licensing, Activation, and Trial Mode

How the trial mode works

© 2011 Inductive Automation

Ignition can be used for 2-hours at a time, with no other restrictions. At the end of the demo period, the system will stop most functions. For example, transaction groups will stop logging, and clients will show a demo screen. By logging into the gateway, you may re-start the demo period, and enable another 2 hours of execution. The demo period may be restarted any number of times.

All portions of the gateway (and therefore, all modules) share the same clock and will timeout simultaneously.

How licensing works

Ignition is a modular platform, and therefore, is licensed based on modules. Licensed and un-licensed modules can operate side-by-side, so some modules can be tested in trial mode while others run in a licensed state.

Despite the modular licensing, each server only has a single **CD-Key** and license file. That is, there is a single license file that dictates which modules are currently activated.

When module(s) are purchased, you will receive a "cd-key" - a six digit code that identifies your purchase. You then use this cd-key to *activate* the software, through the Ignition Gateway. Activation is a process by which that cd-key and its associated parameters get locked to the machine that you are activating. If you are adding an additional module, your account will be updated, and you can re-use your existing cd-key to activate the new features.

It is possible to un-activate your cd-key, freeing it for activation on a different machine.

How activation works

Activation, as mentioned above, is the method by which a cd-key is locked down to the install machine, and the modules are notified of their license state. It is a two step process that can be performed automatically over the internet, or manually through email or through the Inductive Automation website.

Step 1 - Enter CD-Key
When the software is purchased, you are provided with a six digit *cd-key*. After logging into the gateway configuration, go to `Licensing > Purchase or Activate`, and select "Activate". Enter your cd-key.

Step 2a - Activate over Internet
If your computer has internet access, activating over the internet is the easiest option. A secure file will be created with your cd-key, and sent to our servers. The response file will then be downloaded and installed, completing the entire process in seconds.

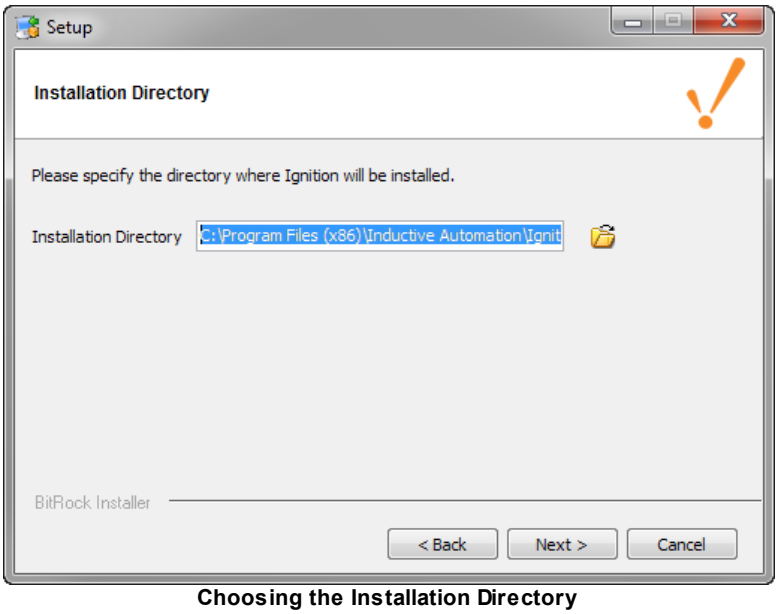
Step 2b - Activate Manually
If you do not have internet access on the installation machine, you must activate manually. In this process, an activation request file is generated (`activation_request.txt`). You must then take this file to a machine with internet access, and email it to support@inductiveautomation.com, or visit our website to activate there. Either way will result in a license file (`license.ipl`) being generated, which you then must take back to the Gateway machine and enter into the License and Activation page.

© 2011 Inductive Automation

1.4 Quick Start

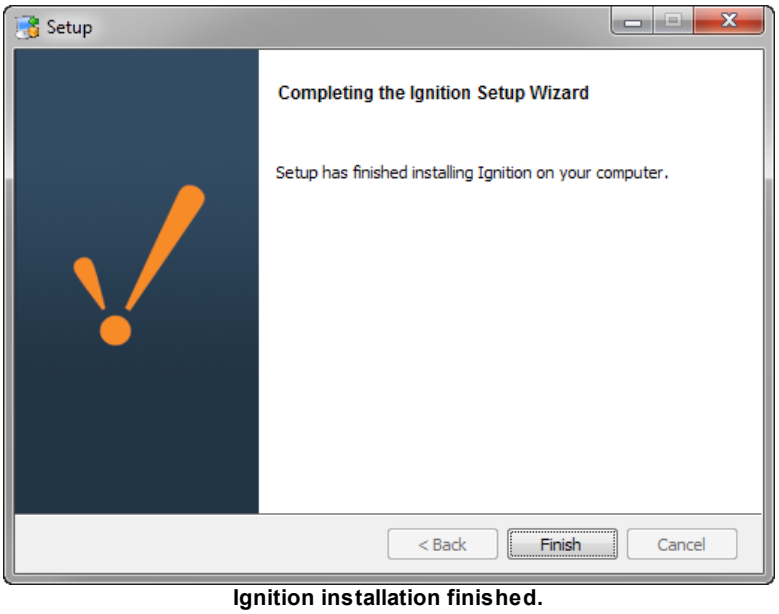
1.4.1 Installation (Windows)

Ignition by Inductive Automation is really easy to install. To get started, simply download the Windows executable installer from our website, and double-click on it. After it starts up, if you agree to the licensing terms, continue on to the next step.



Choosing the Installation Directory

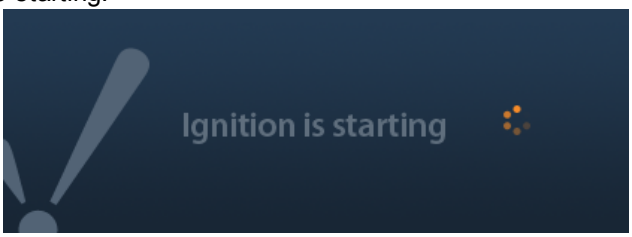
The only option in the installer is to chose where Ignition is installed on your hard drive. The default (your Program Files directory) is usually a good choice. Once Ignition starts installing, it may take a few minutes to finish. Ignition installs itself as a Window Service, so it will start automatically when your computer starts up



Ignition installation finished.

© 2011 Inductive Automation

When the installation is complete, press the "Finish" button. You will see a splash screen informing you that the Ignition service is starting.



Ignition service starting up...

Once the Ignition Gateway starts up, your web browser will open and bring you to the Gateway Homepage.

1.4.2 Installation (Linux)

To install under a Linux OS, it is assumed that you are comfortable operating a shell.

The first step is to download the Linux distribution archive of Ignition from our website. The Linux downloads, which are zip files, are listed underneath the Windows executable installer.

After downloading the Linux distribution archive, follow these directions to install Ignition as a Linux service. You'll also find these directions in the distribution file's README.

All of these commands should be run as *root*. Prefix everything with `"sudo"` or run `"sudo su"` first.

1. **Install Java 6**
   If Java 6 is not already installed, run this command to install it:
   ```
   apt-get install sun-java6-jre
   ```

2. **Unzip files.**
   We're going to install Ignition into /usr/local/bin/ignition. Run the following command, where `<<ignition-linux.zip>>` is the path to the Ignition Linux distribution that you downloaded.
   ```
   unzip <<ignition-linux.zip>> -d /usr/local/bin/ignition
   ```

3. **Switch Directories**
   Change directories into the install directory.
   ```
   cd /usr/local/bin/ignition
   ```

4. **Make Files Executable**
   Execute these two "chmod" commands to make files executable.
   ```
   chmod +x ignition.sh
   chmod +x ignition-gateway
   ```

5. **Create symlink in init.d**
   To turn the application into a service, create a symlink to ignition.sh in your init.d file
   ```
   cd /etc/init.d/
   ln -s /usr/local/bin/ignition/ignition.sh ignition
   chmod +x ignition
   ```

6. **Install the Service**
   Use update-rc.d to install Ignition as a service.
   ```
   update-rc.d ignition defaults
   ```

© 2011 Inductive Automation

7. **Start up Ignition**
   Ignition is now installed a service. It will start up when the computer boots up. To start it up now, use this command:
   ```
   /etc/init.d/ignition start
   ```

That's it! Ignition is now starting up. See the README file for information about how to stop and uninstall the service.

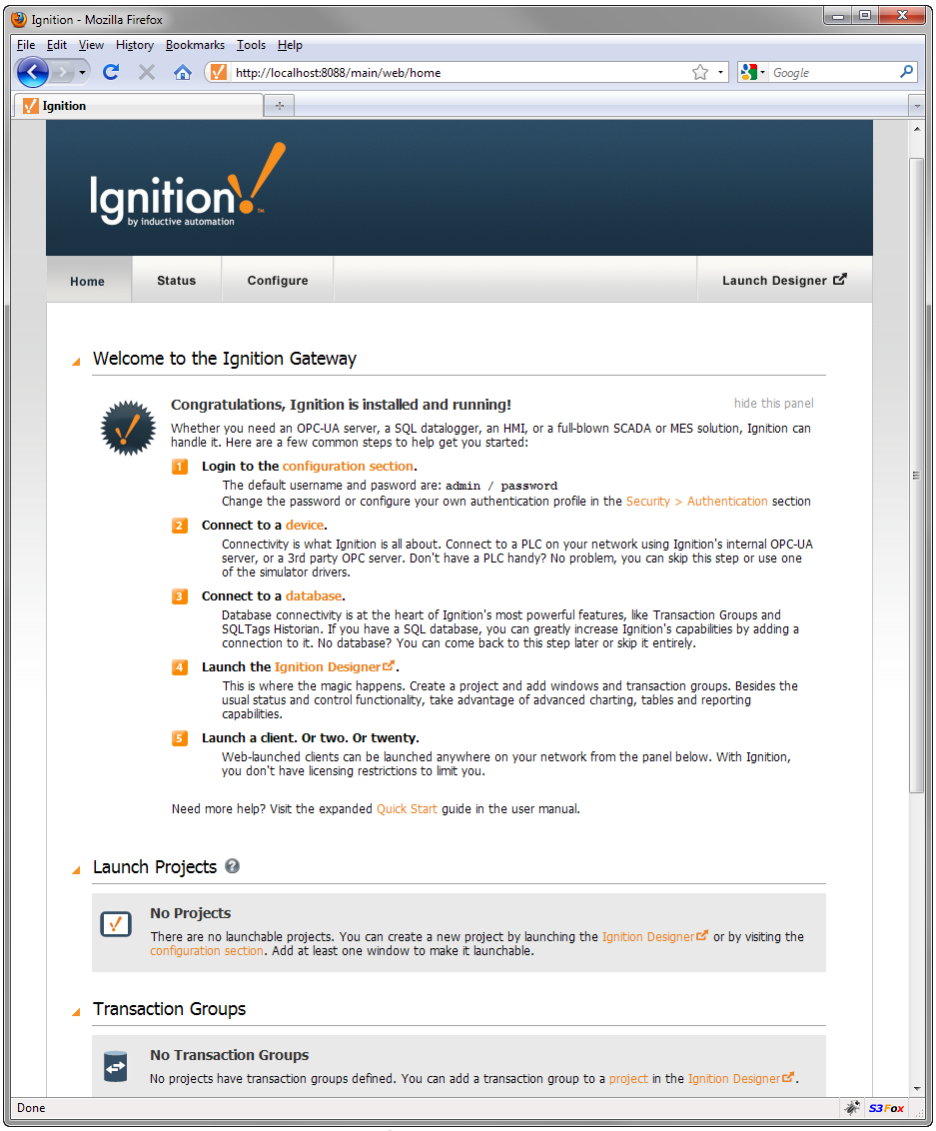
1.4.3 Gateway Homepage

The Ignition Gateway is a web server. When it is running, you access it through a web browser. For example, if you are logged into the computer that you installed Ignition on, open up a web browser and go to the address:

```
http://localhost:8088
```

and it will bring up the *Gateway Homepage*, pictured here.

© 2011 Inductive Automation


Ignition - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8088/main/web/home
Ignition
Ignition
by inductive automation
Home
Status
Configure
Launch Designer
Welcome to the Ignition Gateway
Congratulations, Ignition is installed and running!
hide this panel
Whether you need an OPC-UA server, a SQL datalogger, an HMI, or a full-blown SCADA or MES solution, Ignition can handle it. Here are a few common steps to help get you started:
1 Login to the configuration section.
The default username and pasword are: admin / password
Change the password or configure your own authentication profile in the Security > Authentication section
2 Connect to a device.
Connectivity is what Ignition is all about. Connect to a PLC on your network using Ignition's internal OPC-UA server, or a 3rd party OPC server. Don't have a PLC handy? No problem, you can skip this step or use one of the simulator drivers.
3 Connect to a database.
Database connectivity is at the heart of Ignition's most powerful features, like Transaction Groups and SQLTags Historian. If you have a SQL database, you can greatly increase Ignition's capabilities by adding a connection to it. No database? You can come back to this step later or skip it entirely.
4 Launch the Ignition Designer.
This is where the magic happens. Create a project and add windows and transaction groups. Besides the usual status and control functionality, take advantage of advanced charting, tables and reporting capabilities.
5 Launch a client. Or two. Or twenty.
Web-launched clients can be launched anywhere on your network from the panel below. With Ignition, you don't have licensing restrictions to limit you.
Need more help? Visit the expanded Quick Start guide in the user manual.
Launch Projects
No Projects
There are no launchable projects. You can create a new project by launching the Ignition Designer or by visiting the configuration section. Add at least one window to make it launchable.
Transaction Groups
No Transaction Groups
No projects have transaction groups defined. You can add a transaction group to a project in the Ignition Designer.
Done
S3Fox

The Gateway Homepage with Quickstart Steps

The first time you go to the Gateway Homepage, It will show you 5 common steps to help you get started. You can follow along with these steps and/or with this quick-start guide - they follow the same basic workflow.

The Configuration Section

The first step is to log into the Gateway's *Configuration Section*. To do this, click on the large "Configure" button in the top navigation bar.

You will be asked to log in - the default username/password is: `admin / password`

Once you are in the configuration section you can navigate to the various configuration areas using the menu tree on the left-hand side of the page. Learn more about using Gateway's web interface in the Gateway Navigation section.

© 2011 Inductive Automation

1.4.4 Connect to a PLC

Now that we've installed Ignition and have logged into the Configuration section of the web interface, lets install a *device.* A device is a named connection to an industrial device like a PLC. There are also "simulator" devices that you can add that will mimic a connection to a real device in case you don't have one handy.

This step is optional! You can come back to it later if you'd like. The next steps will be more interesting if you add a device now, however.

These devices are part of the integrated Ignition OPC-UA server module. If you have a *classic* OPC server (OPC-DA 2.0 or 3.0) that you'd like to connect to, see the OPC-COM Module.

Adding a Device

To add a device, use the left-hand side configuration menu to go to the **OPC-UA > Devices** section. Once at the Devices page, click on the **Add a Device...** link at the bottom of the table.

Choose a Driver

You will be given the option to pick the *driver* for the device you want. If you don't have a device that matches one of the available drivers, you can add a simulator device so you have some data to play with.

Configure the Device

After you choose the driver, you'll need to give your device a name and set some options. Typically, the options are just an IP address to connect to. See the documentation for your specific driver for more details.

After configuring the device, simply press the "Save" button to add your device. You can check the connectivity status of your device in the Gateway Status section, under Ignition OPC-UA Server.

1.4.5 Connect to a Database

Many of the advanced features of Ignition, such as the Transaction Groups and SQLTags Historian require a connection to an external database. If you don't have a database, like Microsoft SQL Server, MySQL, or Oracle installed, don't worry - **you can come back to this step later**.

Add a Database Connection

Make sure you are in the Gateway Configuration section of the Gateway's web interface. To connect to a database, use the left-hand side configuration menu to go to the **Databases > Connections** section. Once at the Database Connections page, click on the **Create new Database Connection...** link at the bottom of the table.

Pick a JDBC Driver

Database connections in Ignition are powered by *JDBC drivers*. Ignition ships with drivers for Microsoft SQL Server, MySQL, Oracle, and PostgreSQL. Adding a new JDBC driver for other databases, like IBM DB2, is not very difficult, see Adding a JDBC driver.

Pick the JDBC driver your database, and click on the "Next >" button.

Configure the Connection

Each database connection needs a name, which should consist of letters, numbers and underscores.

© 2011 Inductive Automation

The *Connect URL* parameter is the most important parameter of the connection. This parameter defines where the database server is on the network, and what database to connect to. Each database's connect URL is slightly different. Follow the instructions given for the driver you chose.

The *Extra Connection Properties* field is used less frequently, but is important for some drivers, such as SQL Server's driver. It is a semi-colon separated list of key-value pairs. Each driver has its own set of property keys that it accepts.

The *Username* and *Password* fields are used to supply credentials to the database connection.

For example, suppose we wanted to connect to a database named "ProcessDB" on the server at IP address 10.0.25.122. Here are some examples for the different databases:

Microsoft SQL Server	`jdbc:sqlserver://10.0.25.122\InstanceName`
	with extra connection properties:
	`databaseName=ProcessDB`

MySQL	`jdbc:mysql://10.0.25.122:3306/ProcessDB`
Oracle	`jdbc:oracle:thin:@10.0.25.122:1521:ProcessDB`
PostgreSQL	`jdbc:postgresql://10.0.25.122:5432/ProcessDB`

When you are done configuring your database connection, click on the "Create New Database Connection" button to continue. You can check the status of your database connection in the Gateway Status section under Database Connections.

1.4.6 Launch the Designer

Now that we have some connections set up (or not, if you skipped the last two steps. They were optional, after all), we'll *web-launch* the Designer.

Web-Launching

Web-launching is one of the best parts about Ignition. This is how we launch both the Designer, which is where you'll configure your projects, and our Ignition Vision Clients. Web-launching is a technology that lets you launch a full-fledged application with zero installation just by clicking a link on a webpage. This means that with Ignition, you'll only ever need to *install* the Gateway. All of your Clients and Designers do not need to be installed, and they are always kept up-to-date. Once you start using web-launched clients, you'll wonder how you ever did without them.

In order to successfully web-launch, you'll need Java 5 or Java 6 installed. If you're on the computer that's running the Ignition Gateway, you already have Java installed - the Ignition installer made sure of that. If you're on a computer that is accessing the Gateway over the network, the Java Detection panel on the bottom of the Gateway's homepage will detect whether or not Java is installed.

Launch the Designer

To launch the Designer, simple press the big "Launch Designer" button in the upper right-hand corner of any Gateway page. Once the Designer starts up, you'll see the login pane. The default username for the Designer is the same as for the Gateway Configuration section: `admin` / `password`

The next window will prompt you to open a project. You don't have any projects yet - so click on the " Create New" tab. Enter a name for your new project (no spaces!) and press the "Create" button. That's it - you're now editing your project!

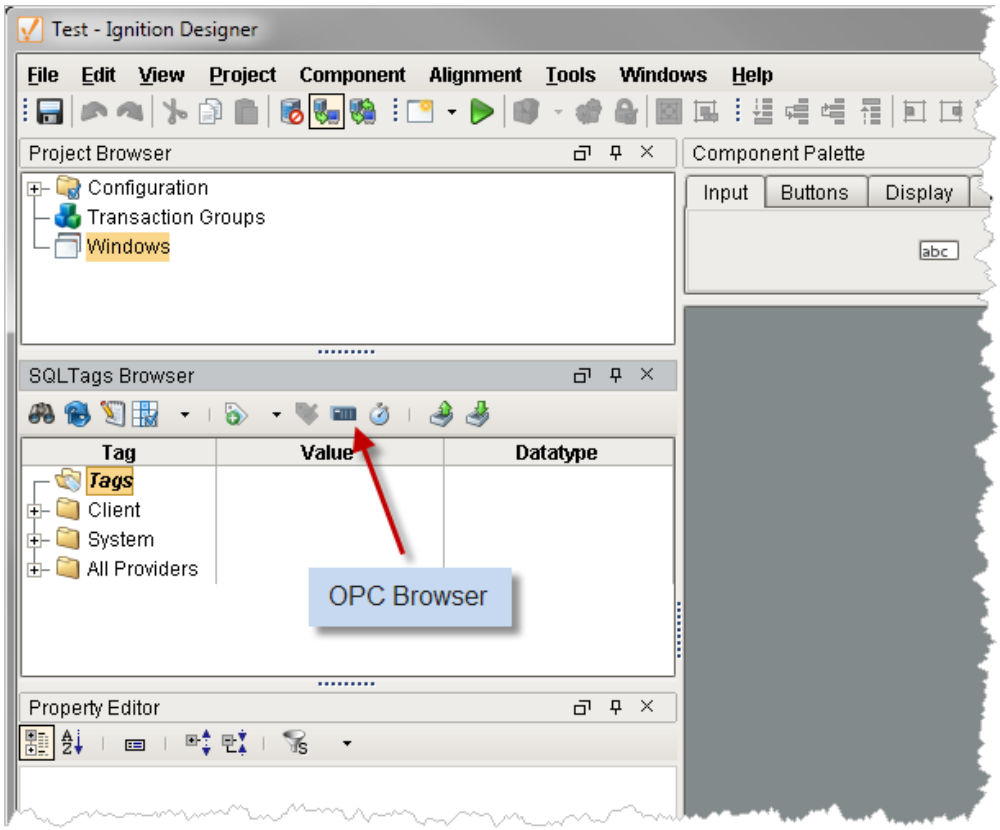
© 2011 Inductive Automation

1.4.7 Create some SQLTags

Once you're in the Designer, a good first step is to create some SQLTags. You'll use these tags for realtime status and control and to store history with the SQLTags Historian.

Drag-and-Drop from OPC

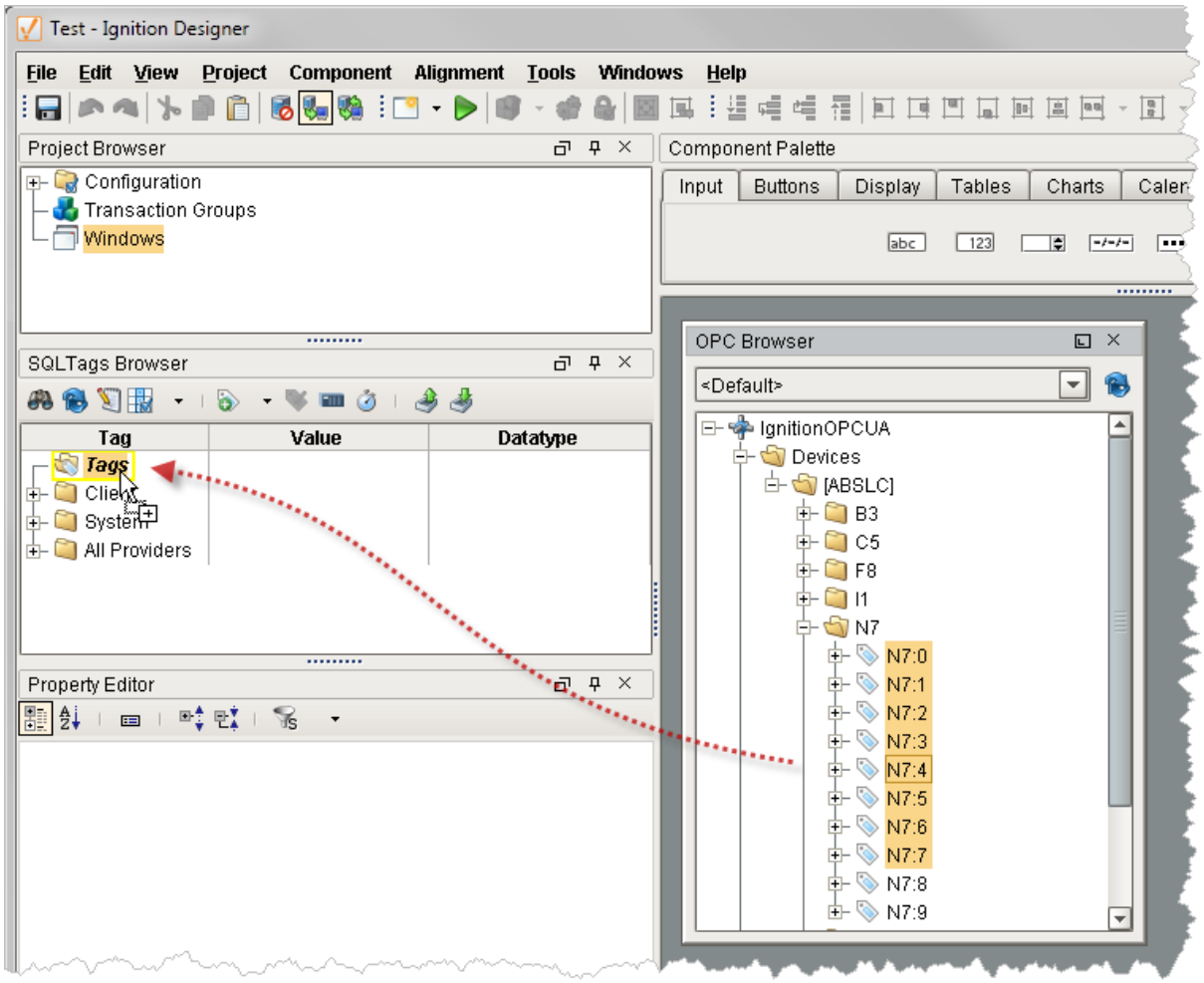
If you created a *device* in the earlier step, the easiest way to create some SQLTags is through drag-and-drop. Select the "***Tags***" folder, and then click on the device icon to bring up the OPC Browser.



The OPC browser button

Now you can browse all of your OPC connections. By default you've got a connection to the internal Ignition OPC-UA server, which has the device in it that you created earlier. Browse the device and find some tags that you're interested in. Highlight the tags and drag them into the "***Tags***" folder in the SQLTags Browser panel.

© 2011 Inductive Automation



Creating SQLTags from the OPC browser

Thats it - you now have some SQLTags. You should see their values come in and start updating.

1.4.8 Create a Window

Lets create a window so we can use our SQLTags for some basic status and control. Click on the **New Window** ( ) icon in the toolbar or use the **File > New > Window** menu item.

SQLTags are used in windows to power property bindings on components. The easiest way to make some components that are bound to SQLTags is to simply drag and drop some tags onto your window.

When you drag a SQLTag onto a window, you'll get a popup menu asking you what kind of component to make. You can *Display* the tag with some components, and *control* the tag with other components. Drag a few tags onto the screen to experiment with the different options.

As you're editing your project, you can hit the Save ( ) to save you changes. In Ignition, you're not editing a *file*. Your Designer is linked up to the Ignition Gateway. When you hit save, the project is saved back on the central Gateway. Any running Clients would be notified that there is a new version of the project available.

See also:
Creating Components / SQLTags Drag-n-Drop

© 2011 Inductive Automation

1.4.9 Launch a Client

Now that we've created a window, lets launch a client to see it in action. Make sure you've saved your project, and then go back to the Ignition Gateway homepage. Your project will appear in the Launch Projects panel with a big **Launch** button its right. Click on the launch button to start up a Client.

You'll need to log into the Client. By default, a new project uses the same authentication profile as the Gateway - so the `admin` / `password` credentials will work.

Once you've logged in, you will see your window running. Now go back into the Designer and make a change to the window and hit **Save**. Your Client will show a notification that there are updates to the project. Click on the notification and the Client will update itself.

Thats it - you can launch as many clients as you want! Try it out - if you've got other computers on the same network as the Gateway computer try launching on them too. Make sure that your Gateway computer doesn't have a Firewall enabled, or if it does, it is allowing traffic on port 8088 - the default port for the Ignition web server.

See also:
Web Launching

1.4.10 Create a Transaction Group

Transaction groups are used to store history, log events, synchronize databases tables with PLC, perform calculations, and many more data-centric tasks. To get started, let's create a basic History Group and start logging some PLC values to your database.

Switch the Designer to the Transaction Group *workspace* by clicking on "Transaction Groups" in the Project Browser panel.

Make a new Historical Group by clicking on **File > New > New Historical Group**. Your new group, named "Group" will be selected. The OPC-browser panel is now *docked* to the lower left side of the screen. Browse your OPC device again and drag some OPC tags into the **Basic OPC/Group Items** section.

Your group starts out *disabled*. To enable logging, press the **Enabled** button above the item tables. You can also change the **Table Name** field under the **Action** tab to name your table something interesting.

Right now your group only exists in the Designer - you need to **Save** the project to **start the group**. Groups execute on the Ignition Gateway. Save your project now.

Your group is now running, logging data to your database connection. To see the data, you can use the Ignition Designer's built-in database query browser. The easiest way to do this is to click on the Query Browser ( ) button to the right of your group's Table Name field. The Query Browser is a convenient way to directly query your database connection without leaving the Ignition Designer. Of course - you can also use any query browser tools that came with your database.

© 2011 Inductive Automation

Overview

Part II

2 Overview

2.1 What is Ignition?

Ignition is an *Industrial Application Server*. Installed as server software, it uses webpages and web-launching to create a wide variety of industrial applications. These sorts of applications typically fall under the definitions of HMI, SCADA, and MES applications. Ignition achieves its functionality through a modular architecture, meaning that multiple pieces work together seamlessly to provide features like:

OPC-UA Server
OPC-UA the leading industrial standard for data access. Using the OPC-UA Module, Ignition will act as an OPC-UA server, serving data collected by its built in drivers to other Ignition modules, as well as third-party OPC-UA clients.

For more information about OPC, see the section *What is OPC?*

For more information about the device drivers available in Ignition, see *About Ignition Device Drivers*

Data Logger
Ignition offers robust data-logging functionality. The SQL Bridge module offers historical logging, trigger based transactions with handshakes, and much more. Additionally, the ground-breaking *SQLTag Historian* feature makes it easier than ever to store and use historical process data.

Status & Control
Ignition offer first class status and control functionality, and can be used to create single-user terminals as well as distributed systems. *SQLTags,* Ignition's tag system, provides many powerful features and unparalleled ease of use. By simply dragging-and-dropping, you can create a powerful status and control screen in minutes. Features such as clustering and *Panel Edition* licensing help create dependable, fault-tolerant systems.

Alerting Server
Flexible alert monitoring is built into SQLTags, and the Ignition gateway supports a variety of logging and notification features. *Alert Distribution Groups* allow you to send email alerts with a high level of control. Alert history can easily be stored and queried, making it easy to track and analyze common problems in your process.

Data Analysis
Ignition offers industry-leading trending and data analysis functionality. The power of SQL database access is built in from the ground up, and offers a tremendous amount of power in today's IT centric plants. Powerful charting, tables, and reports combined with Ignition no-install, web-launched distribution model offer new possibilities in data analysis.

PDF Reporting
Create dynamic, data-rich PDF reports using the Reporting module. Leveraging the power of SQL databases, it's easy to tie together production and business data.

See Also:
Modules

2.2 Architecture

2.2.1 Architecture Overview

Ignition is a powerful server application that consists of many parts. However, it is designed to be approachable and easy to start using up front, with the power to accomplish many advanced tasks as the user requires them.

© 2011 Inductive Automation

In order to effectively use this guide and to get started, there are a few basic concepts about the architecture of Ignition that should be understood from the start. These key concepts are located in the *System Concepts* chapter.

In addition to the internal architecture of Ignition, there are many *system architectures* that are possible. This is how Ignition is installed, and how it interacts with other key systems, such as Databases and OPC servers. The *Architecture Diagrams* chapter outlines a variety of different possibilities. Most users will begin working with Ignition using a standard architecture, where the software and all components are all installed on a single machine. To receive the full benefit of Ignition, however, it's important to know what is possible- and therefore it is recommended that you at least browse through the various architecture diagrams and advanced architecture topics. As your system expands, you can come back to investigate the possibilities in more depth.

2.2.2 System Concepts

2.2.2.1 Ignition Gateway

The Ignition gateway is the primary service that drives everything. It is a single application that runs an embedded web server, connects to data, executes modules, communicates with clients and more.

Starting and Stopping the Gateway

After installation, the gateway will be started automatically. Manually stopping and starting the service will depend on the platform that you are using.

Windows

Access the service control utility through `Control Panel>Administrative Tools>Services`. The "Ignition" service entry can be used to control the run state of the gateway.

Linux

It is possible to control the service through the *Ignition.sh* script. It can be called with the `start` and `stop` parameters to perform the relevant operations.

For example:

```
>./Ignition.sh start
```

Access the Gateway

To monitor and configure the gateway, you will use a web browser to connect to the web server operated by Ignition. See the Gateway Navigation section for information about how to connect, and a description of the gateway.

Gateway Control Utility

The Gateway Control Utility is an application that can be used to monitor the gateway and perform high-level tasks that aren't available inside of the web application. The GCU can be found from the Start menu in windows and in the install directory in other platforms. See Gateway Control Utility under the Basic Usage chapter for more information.

2.2.2.2 Ignition Designer

The Ignition Designer is a web-launched application that lets you configure and build your projects. The application is launched from the gateway homepage. See Gateway Navigation for more information.

© 2011 Inductive Automation

Project structure and the designer

The Ignition gateway runs projects, and it is possible to create any number of projects. Each project consists of settings and *project resources*, objects that are defined by modules.

When the designer is opened, you will be asked to select or create a project. The designer will then display one or more *workspaces* according to the modules that are installed, and you will be able to create and modify different types of project resources. When the project is saved, the modifications are sent to the gateway, where they are handled by the appropriate modules.

Working concurrently on projects

It is possible for multiple designers to operate on the same project concurrently. This situation is common in large projects where multiple people may be involved. When a particular resource is being edited, it will be *locked*, and the other designers won't be able to edit it. When the project is saved, the resource will be unlocked.

Concurrent edits will be received by other designers only when "Update Project" is clicked from the file menu.

2.2.2.3 Ignition Vision Clients

The web-launched Vision Clients are the "runtimes" of the Vision module. They run as full applications and feel like a traditional installed client, without the need to install and manually synchronize projects.

Launching clients

Clients are launched from the Gateway homepage, for a specific project. See the Gateway Navigation section for more information.

Updating client projects

Clients are automatically notified when project updates are available. When launching a client, the most recent version of the project will always be used. If an update occurs while a client is open, a bar will appear across the top of the client, notifying the user. By clicking on the bar, the new project modifications will be downloaded and applied. You can also configure a client to use *Push* notification, which means that the new version will be downloaded and applied automatically.

2.2.2.4 Mobile Clients

With the Mobile Module installed, you can launch your same Vision projects on any modern smartphone or tablet. This ability does not require any re-design of your projects - a mobile client launches the same projects that the Vision clients launch.

How it works

Normally, you can't launch Vision Module projects on mobile devices. This is due to the technical limitation that Java SE (Standard Edition) does not run on mobile devices. The Mobile Module gets around this limitation by launching the client on the Gateway in a special *headless* (invisible) mode, and then using HTML5 and AJAX to show the client's screen on the mobile device's browser.

© 2011 Inductive Automation

Networking

Typically, the mobile device will connect to the Ignition Gateway via the facility's wireless LAN (802.11) infrastructure. To launch a mobile client, the mobile device simply connects to the Ignition Gateway by pointing its web browser to the Gateway's LAN address. It is important to understand that normally, the traffic is not going over the device's cellular connection. This wouldn't work, because the cellular connection connects to the internet, and without explicit setup, an Ignition Gateway is not accessible from the outside internet.

Remote (as in, beyond the reach of 802.11 wireless LAN) mobile access can be enabled through the same networking strategies that enable remote access for standard Vision clients. Somehow, the mobile device must be able to access the Ignition Gateway via its cellular connection. One strategy would be to set up a VPN router and configure the mobile device as a VPN client. This way, the mobile device could directly access the LAN address of the Gateway as if it were on-site. Another technique would be to put the Ignition Gateway in a DMZ so that at least one NIC had a public IP address. Or, an edge router could be configured to port-forward the HTTP and HTTPS ports to the Gateway. Coordination with your I.T. department would be advised when attempting to set up remote access.

2.2.2.5 Database Access

Access to relational databases is at the heart of the Ignition platform. Ignition can connect to any SQL database that has a JDBC driver, though depending on the database's capabilities, some features may not be available.

Why use Databases?

Ignition can perform many tasks without the use of a database. For instance, the Vision and OPC-UA modules can be used to create powerful HMI status and control screens, SQLTags can be used to generate alarms that can be sent over email, and more. However, tightly integrated database access is a key feature that makes Ignition stand out from its competitors.

Modern relational databases offer amazing storage and querying capabilities with great performance at a price that is incomparable to older legacy historians. While it is true that historians still have a place in the industry, for most applications relational SQL databases not only suffice, but offer much more than what was previously available. Using SQL, you can store and track production information with ease. However, you can also correlate that data to who was on shift, previous runs, downtime, inventory levels and more, naturally and easily. Make the data available to more people using the Vision module's web-launch clients, or integrate the data directly into your company's internal or external website. SQL databases are at the heart of the web and modern corporate IT systems, and now thanks to Ignition, the plant floor as well.

Configuring Database Access

See the Database section under Gateway Configuration for more information about connecting to databases through Ignition.

2.2.2.6 OPC-UA

OPC-UA is the latest revision of the OPC specification, which offers platform and vendor neutral transfer and use of industrial data. The specification plays a crucial role in Ignition, and is the primary data access specification used in the Gateway. Ignition supports connections to any number of OPC-UA servers created by any manufacturer, provided that they are compliant to the specification. The data is then used to drive all aspects of the system. Creating connections to OPC-UA servers is described in the Gateway Configuration section.

© 2011 Inductive Automation

Creating Distributed Systems with OPC-UA

OPC-UA breaks down boundaries and enables free data flow. Using standard TCP/IP instead of legacy DCOM, OPC-UA makes it easy to securely transfer data between networks and though firewalls. All OPC-UA connections are based on the same technology, which means that a connection to your local machine is not entirely different than a connection to a machine that's far away. This enables the creation of highly distributed system, and in combination with other features of Ignition can lead to much more connected enterprises.

For example, imagine a corporate network with an office in the center, and remote processes connected through a VPN, which would pass through a variety of connections. Each remote site could have an Ignition installation running only an OPC-UA module that would report data back to a central facility and record it in a database. The overall system cost would be very low, the data could be managed centrally in a single location, and then made available to all interested parties through the Vision module or any application that could access the database.

2.2.2.7 SQLTags

Introduction

SQLTags™ is the tag database mechanism of Ignition. Each tag in Ignition is a SQLTag, irregardless of whether the value comes from OPC, an expression, or is static. SQLTags provide a variety of configuration options, such as alerting, scaling, and historical storage.

SQLTags are stored in *tag providers*. By default, a fresh Ignition installation will have an *internal* tag provider - this can be thought of as a standard internal tag database. Additionally, it is possible create *external* DB-based tag providers, thus turning your SQL database into the tag database. This ability opens up some very flexible architectures and is the primary reason why SQLTags have their name.

Main benefits of SQLTags

SQLTags work naturally with Ignition to offer many exciting features:

Drag and Drop screen design
Using the Vision module, drag and drop tags onto a window to automatically create new bound components. Drag tags onto existing components or properties to quickly bind them to the data. Creating powerful status and control screens is literally just clicks away!

Performance and Scalability
SQLTags offer terrific performance on both the gateway and client side. On the gateway side, the system can support thousands of value changes per second, and hundreds of thousands of tags overall. On the client side, SQLTags improve efficiency with their lightweight subscription architecture. Adding additional clients creates a nearly negligible effect on the database and gateway performance.

Integrated component feedback
SQLTags feature a *quality* and *overlay* system for Vision module components. If a tag's data quality is anything but good, a component that depends on it will get a visual overlay. Input components display an animated overlay while write pending requests are being written. These features effectively communicate the status of the system at a glance.

One-click historical logging with SQLTags Historian™
SQLTags Historian makes it easier than ever to store and use historical data. By simply selecting a checkbox on a tag, historical data will be stored in an efficient format in your SQL database, and will

© 2011 Inductive Automation

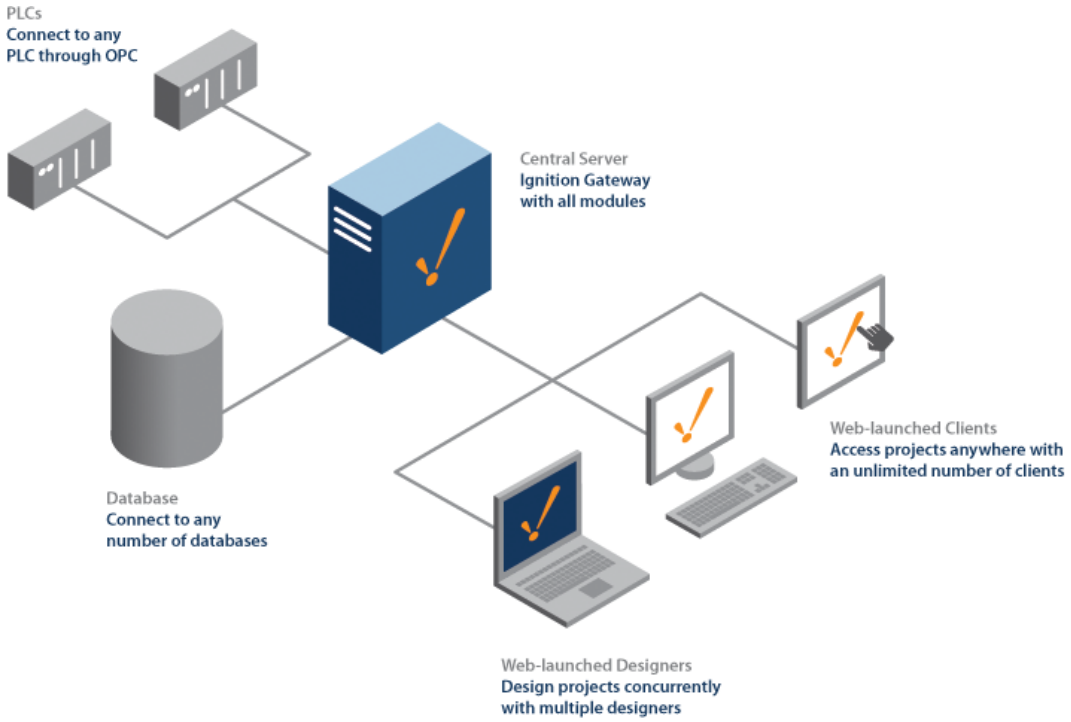
be available for querying through scripting, historical bindings and reporting. Drag-and-drop tags directly onto an EasyChart™ to create trends, or onto a table to display historical values. SQLTags Historian's robust querying features give you great flexibility in how you retrieve the data.

Getting started with SQLTags

See the SQLTags section under Project Design for more information about creating and using SQLTags and SQLTags Historian.

2.2.3 Architecture Diagrams

2.2.3.1 Standard Architecture

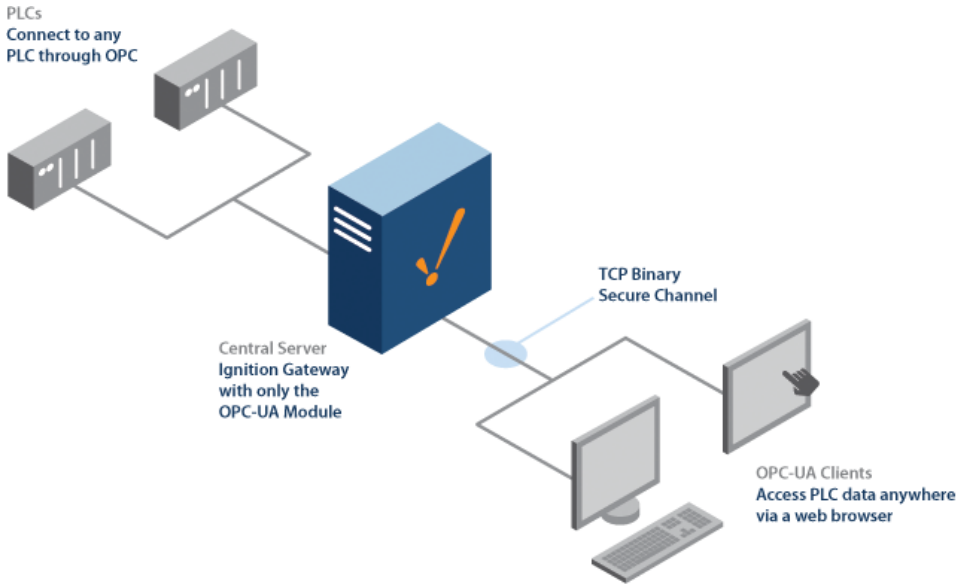


In the standard architecture, a single Ignition gateway is installed on a central server with all of the desired modules. Devices are connected over the network or serial links, and are accessed through Ignition OPC-UA or other OPC servers installed on the same machine. Database connections are made to database servers installed on the same machine or elsewhere on the network.

Any network enabled device with Java and access to the server can launch clients by going to the gateway homepage. Designers can also be launched over the network. Both clients and designers can be launched locally at the server as well.

© 2011 Inductive Automation

2.2.3.2 OPC-UA Architecture

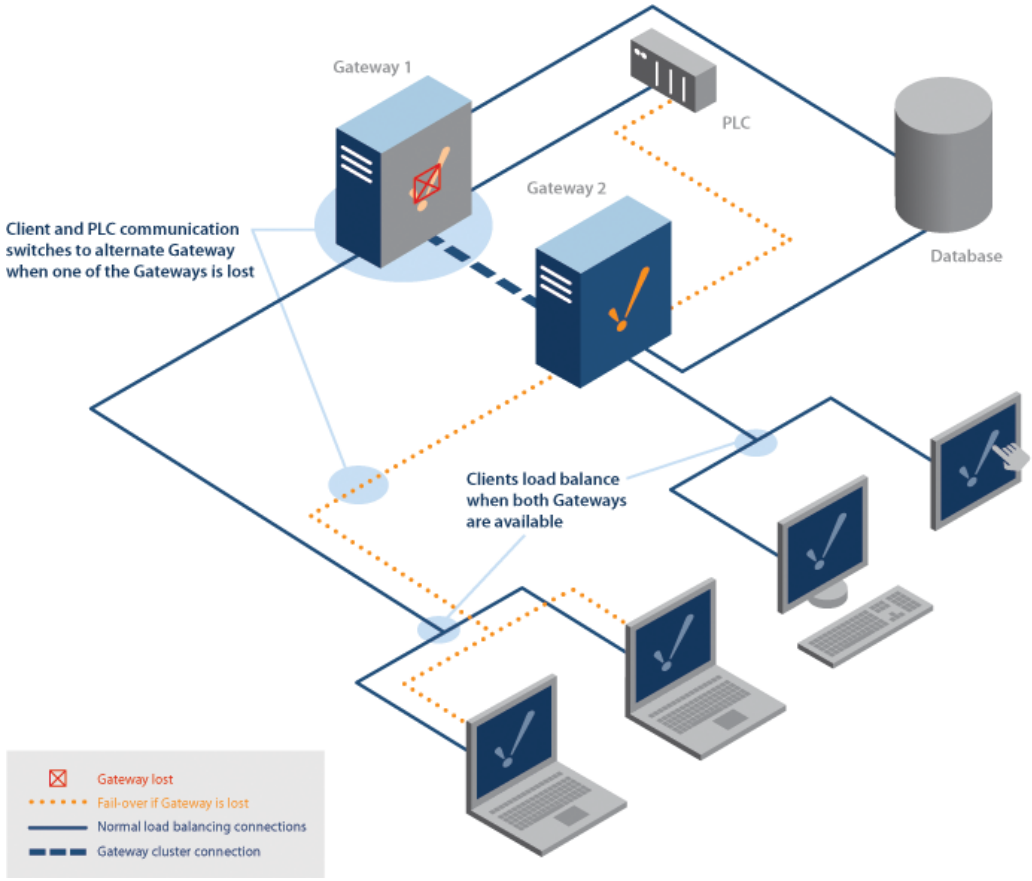


The OPC-UA architecture is very similar to the Standard architecture, but with only the Ignition OPC-UA module installed on the server. In this configuration, the Ignition gateway acts as a dedicated OPC-UA server. Any remote OPC-UA client, including other Ignition gateways, with network access can connect to the server and read and write data.

This installation is useful for aggregating data from many sites. The low installation cost and the secure, painless connections provided by OPC-UA make it easy to access and collect data that wasn't previously available on the network.

© 2011 Inductive Automation

2.2.3.3 Clustered Architecture

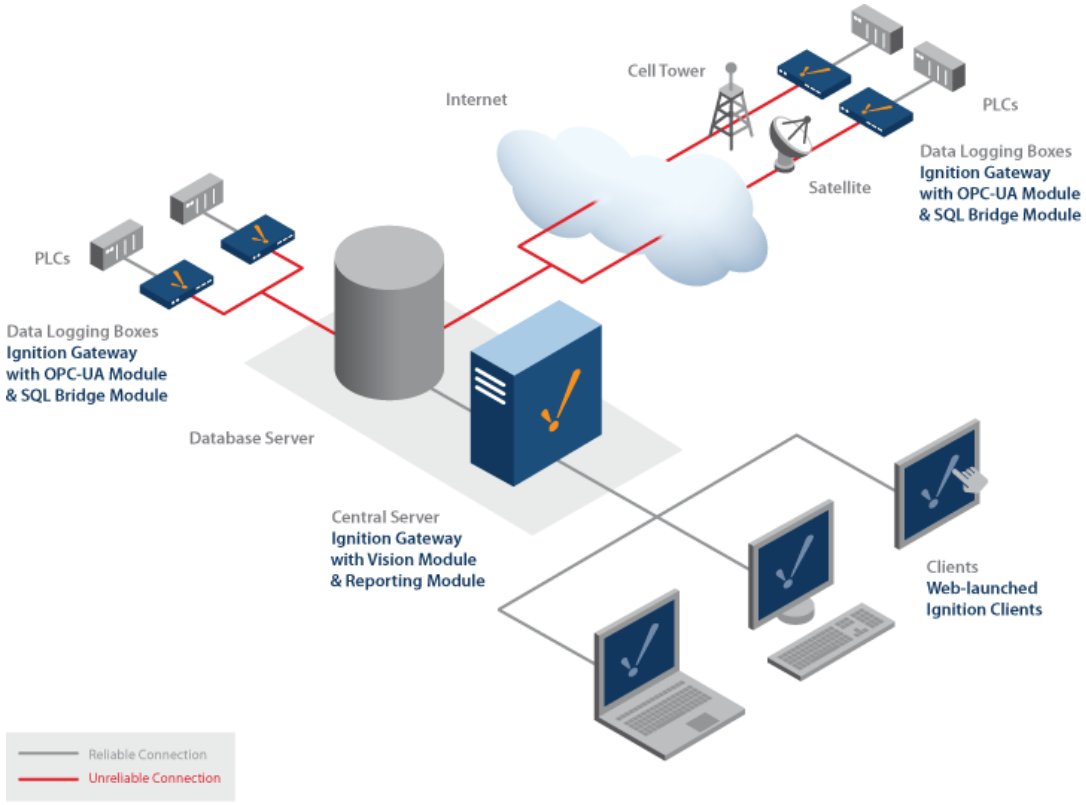

Gateway 1
PLC
Gateway 2
Database
Client and PLC communication switches to alternate Gateway when one of the Gateways is lost
Clients load balance when both Gateways are available
Gateway lost
Fail-over if Gateway is lost
Normal load balancing connections
Gateway cluster connection

In a clustered architecture, two Ignition installations are connected together with the clustering feature to create a dynamic, fault-tolerant system. In addition to fail-over when a server goes down, clients will automatically load-balance between servers, distributing the work load evenly.

Clustering ensures that the projects are synchronized, and can be expanded to include more than two nodes. See Clustering under advanced architecture topics for more information.

© 2011 Inductive Automation

2.2.3.4 Remote Datalogging Architecture



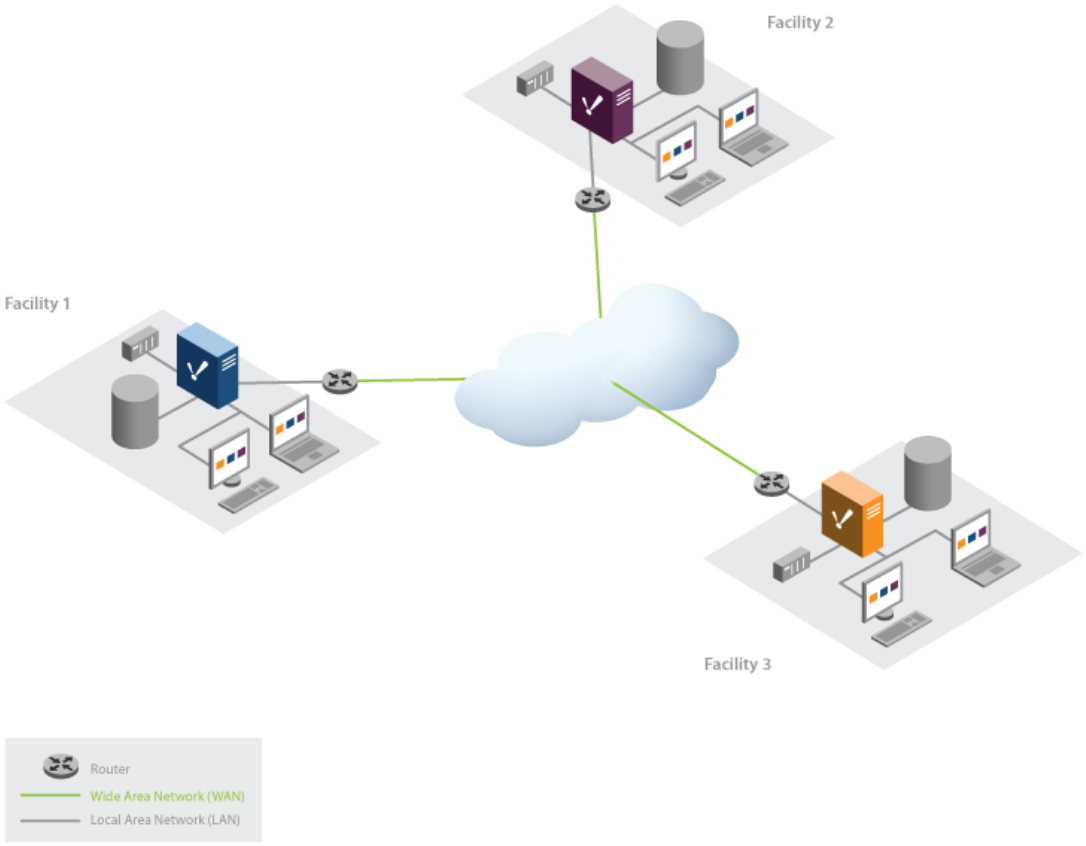
Ignition is highly network centric, with the ability to connect to remote databases and OPC-UA servers as naturally as to local ones. This fact, combined with the built-in *store & forward engine,* make it possible to create wide, geographically dispersed systems with little additional work.

Remote Ignition gateways with the OPC-UA and SQL Bridge modules can store data to central servers. Should the connection go down, the data will be cached until the connection is again available, ensuring that nothing is lost.

Web-launched clients can be used on any computer with access to the network- even over a WAN (wide area network) or VPN (virtual private network). In this way, users can securely access data that has been pulled together from a wide variety of sources.

© 2011 Inductive Automation

2.2.3.5 Wide-area SCADA Architecture

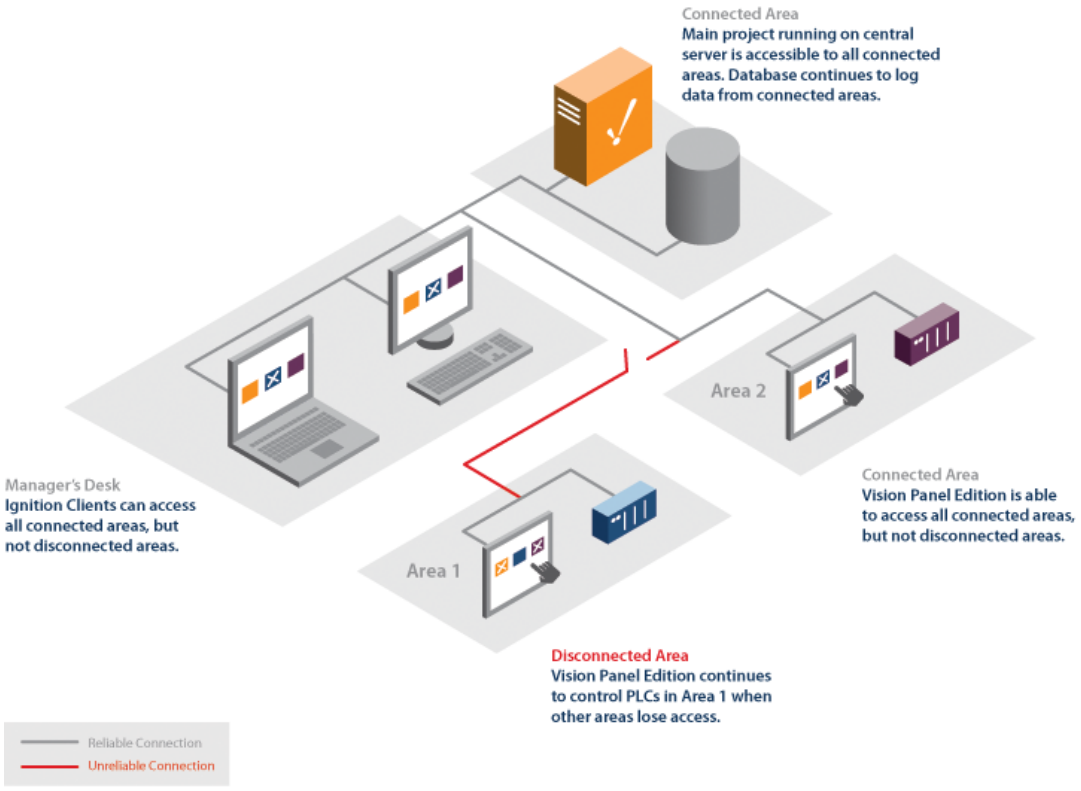


As described in the Remote Datalogging section, the network-centric nature of Ignition makes it easy to access data across a wide area network. Additional key features such as *retargeting* make it possible to blend complete systems hosted at different locations into one seamless architecture.

Each location operates independently, but when combined with a secure inter-location network (such as a VPN over the internet), any location can securely interact with the other locations. There are many possible layers of security, included encrypted communication and the ability to adjust authentication access for each location. For example, users from remote sites may be allowed to only view data, and not modify or control machinery. Conversely, if desired, a central operator may be allowed to control aspects of each location.

© 2011 Inductive Automation

2.2.3.6 Panel Edition Architecture


Connected Area
Main project running on central server is accessible to all connected areas. Database continues to log data from connected areas.
Area 2
Connected Area
Vision Panel Edition is able to access all connected areas, but not disconnected areas.
Manager's Desk
Ignition Clients can access all connected areas, but not disconnected areas.
Area 1
Disconnected Area
Vision Panel Edition continues to control PLCs in Area 1 when other areas lose access.
Reliable Connection
Unreliable Connection

With Ignition Panel Edition, you can install dedicated control clients close to hardware, ensuring availability should the network go down. Using *Retargeting,* the Panel project can be seamlessly integrated in to a larger system, and accessed from remote clients.

2.2.4 Advanced Architecture Topics

2.2.4.1 Clustering

Ignition supports the clustering of two or more gateways together, creating a network of systems that share the same configuration, balance the work of processing client requests, and negotiate the execution of tasks.

Cluster terminology

Node or Peer
A member of the cluster. This is an Ignition Gateway which has clustering enabled and is configured to point to other nodes.

Master
The node currently in charge of the cluster. This node will coordinate other nodes, and is the authority on the current state of the system configuration. It will also execute tasks that can only be run on one node at a time, such as executing transaction groups.

Member
A non-master member of the cluster. Retrieves configuration updates from the master, and handles clients that have been transferred to it.

© 2011 Inductive Automation

System configuration in a cluster setting

The *system configuration,* consisting of projects and settings, is shared across all nodes of the cluster. The master node is the authority, and all nodes who join the cluster will received their configuration from it. The member node's configuration will be overwritten when it joins a cluster, so special care must be taken when setting up clustering between established systems.

Execution of modules in a cluster

Each module will potentially operate differently in a cluster. Some may choose to balance work across the nodes, others will only execute on the master.

Vision module
Clients will be loaded balanced across cluster members. As new clients are launched, they will be transferred to nodes with fewer currently connected clients.

SQL Bridge module
The transaction groups will only be executed by the master.

OPC-UA module
Subscriptions will be monitored across all nodes, but only the master node will communicate with the device, assuming that a clustered connection type is used. If a direct OPC-UA connection is used, each node will communicate with the device.

SQLTags
The tags will be executed independently on all nodes. As mentioned above, if the clustered OPC-UA connection type is used, the data from OPC will be shared across all nodes.

SQLTag Historian
Only the master node will store history to the database.

2.2.4.2 Vision Panel Edition

The Vision Panel Edition is a licensing level that restricts the Vision module to one client on the local system, with tag read & write capabilities. The panel edition cannot access the database or SQLTags Historian.

Uses of the Panel Edition

The Panel Edition is designed to provide an extremely cost-effective way to ensure that local control is available when the network or server goes down. Previous to the introduction of the Panel Edition, touch-screen computers would web-launch clients from the Ignition server. If the network link was unavailable, the client would cease to function. With Panel Edition, the Ignition gateway runs directly on a computer close to the hardware being controlled.

Panel Edition nodes run their own projects, but can be integrated together and with a central server by way of the *Retargeting* feature, in which an Ignition Client running one project can switch to running another project (even on another Gateway) seamlessly.

Restrictions of the Panel Edition

One local client - can only be launched from the gateway machine.

© 2011 Inductive Automation

No database access
No historical access

Exceptions to the restrictions

When a client launched from a fully licensed Vision gateway retargets to a Panel Edition gateway, it confers all of the capabilities of the full system. In other words, the above restrictions do not apply for clients launched on licensed gateways and transferred to the panel. Obviously, some care must be taken in designing projects to support dual-mode operation, such as running SQL queries. While the queries will work when run from a re-targeted client, they will fail on the Panel client. One way to support this is to create multiple projects on the Panel Edition gateway that separate status and control from historical access. You can retarget to either from the full gateway, and then only use the control project from the panel.

Installing Panel Edition

Panel Edition is simply a standard Vision module installation, activated with a CD-key that confers the Panel Edition license level.

2.2.4.3 Remote Logging

The network-centric nature of Ignition offer a large amount of flexibility for building highly distributed systems. One common task is to gather data from remote sites and record it centrally, for easy sharing and additional analysis. There are several ways to accomplish this in Ignition.

OPC-UA Only

OPC-UA is a network-based specification, and is ideal for collecting data from remote locations. Installing Ignition with only the OPC-UA gives you the ability to connect easily and securely from any number of other Ignition installations, or with other OPC-UA clients.

This method only exposes data, however, and the client side must then record it if historical data is desired. If the connection goes down, data will not be available. This method offers the lowest cost, and is suited for situations where the data is not highly critical or historical- for example, remote realtime monitoring.

Remote SQL Bridge

By installing the SQL Bridge module in the remote gateway, you benefit from the *store & forward system* to ensure that no historical data is lost. The system may still be access through OPC-UA as outlined above, or database-based SQLTags may be used to communicate current status. By doing this, it is possible to use SQLTags Historian and alarming, both of which utilize the store & forward system to avoid data loss. This method is ideal for historical data.

2.2.4.4 Distributed SQLTags

SQLTags offer a number of different configuration options. By default, SQLTags are driven by Ignition and stored internally. However, using the Database SQLTags provider, it is possible to store SQLTag configuration and values in an external database. This database can hold tags from multiple Ignition gateways, and each of those gateways will be able to access the tags driven by the others.

Using this methodology it is possible to aggregate multiple remote sites and built a cohesive system

© 2011 Inductive Automation

that spans multiple parts of a single plant, or multiple separate plants entirely.

2.2.4.5 Client Retargeting

Client Retargeting is the method by which Clients running a particular project switch to a different project on the fly, even if the other project is hosted on a different Ignition Gateway.

Retargeting is a key feature used to build distributed systems. It allows you switch between projects and servers as easily as switching between windows. Using Retargeting, even geographically dispersed projects can be presented as a single cohesive unit.

Using Retargeting

Retargeting is accomplished through scripting, usually as a response to a button press or other component event. The `system.util.retarget` function allows you to specify a Gateway and project to retarget to. Authentication will be transferred with the request, and the switch will only occur if the current user also has rights to the target project.

2.3 Modules

2.3.1 Overview

What are modules?

Modules are applications that are built on the Ignition platform and integrate into the platform in order to offer functionality. Most of the main features of Ignition are actually provided by different modules such as the Vision and SQL Bridge modules.

Modules integrate seamlessly into the system and provide things like new designer workspaces, new gateway settings, new drivers, and much more.

Why Modules?

The modular architecture of Ignition offers a wide array of benefits.

Flexible licensing - only license the modules that you need, saving money and reducing complexity compared to big monolithic applications that try to do everything. At the same time, the modules have been designed to offer a broad swath of functionality, to avoid having too many pieces.

Hot-swappable - Modules can be dynamically loaded and unloaded, allowing you to install, remove and upgrade them without affecting other parts of the system. This can have huge implications for big projects where up-time is important.

Increased system stability - Building modules on a common platform means fewer bugs, better isolation, and all around increased stability.

Types of Modules

Module Name	Description
OPC-UA Module	Provides OPC-UA server functionality and an open device driver API.
SQL Bridge Module	Offers transactional datalogging, bi-directional OPC-to-DB synchronization, stored procedure support and more.
Vision Module	Provides HMI/SCADA functionality with web-launched clients.
Reporting Module	Works with the Vision module to provide robust reporting capabilities.
OPC-COM Module	Allows Ignition to connect to older COM based OPC-DA servers.

© 2011 Inductive Automation

2.3.2 OPC-UA Module

The Ignition OPC-UA module offers OPC-UA server functionality with a variety of device drivers and a robust, open driver API.

OPC-UA Server Functionality

This module turns Ignition into an OPC-UA server, capable of handling connections from any OPC-UA compatible client.

Built-in Device Drivers

The OPC-UA module offers a number of device drivers for common protocols out-of-the box, and is easily expandable thanks to the hot-swappable module architecture in Ignition. New drivers can be downloaded and installed in minutes without requiring a system restart or otherwise affecting other parts of the Ignition platform.

Cluster Support

The OPC-UA module ties into the Ignition cluster in order to provide efficient access to device data along with failover redundancy, with no additional configuration.

Public Driver API

Anyone can create new drivers thanks to the open driver API, and users can download and install drivers created by other developers. This is the first time such an API has been made publicly available for a product like Ignition. For more information about creating drivers for Ignition, visit the Inductive Automation website.

2.3.3 SQL Bridge Module

The SQL Bridge module is a robust and flexible tool to map between OPC data and Database data. Different types of Transaction Groups allow you to configure a variety of behaviors, from trigger based historical logging, to bi-directional synchronization, to recipe management and more.

Services provided by the SQL Bridge Module

Multiple types of Transaction Groups that offer:

- Historical logging
- Status updating
- Transactional logging
- DB-to-OPC synchronization
- Stored procedure support

Easy configuration using the web-launched designer

SQLTags Historian

External SQLTags driving

See also:
Transaction Group Overview

© 2011 Inductive Automation

2.3.4 Vision Module

The Vision module provides the visual elements of Ignition. Vision offers a wide range of functionality, and can be used to create HMI style control systems, data analysis and trending applications, executive dashboards, and more. The projects are designed using the Ignition Designer, and clients are web-launched with zero installation from any Java capable computer.

Create your ideal control system in minutes

Combined with the power of SQLTags, it's never been easier to build effective status and control systems. Drag and drop tags on the screen to create automatically bound buttons, HOA toggles, LED displays, value entry fields, and more. Drag tags directly onto component properties to bind bi-directionally in seconds. The innovative *overlay system* provides intuitive data quality feedback with no additional configuration.

World-class charting capabilities

The Ignition Vision module offers a variety of charting and trending options. The *Easy Chart*, as its name suggests, makes it incredibly easy to create useful and powerful charts. The charts support multiple axes, sub-plots, many pens, and hundreds of thousands of data points. Using SQLTags Historian, creating charts is as simple as drag-and-drop, and charts intelligently pull just the data they need, making clients more efficient.

Integrated database connectivity

The Ignition Vision module is the world's most database friendly HMI/SCADA application. Working with SQL databases is integrated into many aspects of the project design process, allowing you to integrate process and business data effortlessly.

Unlimited potential

Web-launched clients, the ability to seamlessly connect multiple projects through Retargeting, and no licensing restrictions on screens, tags, components or clients means the system can grow over time.

2.3.5 Reporting Module

The reporting module is different from other modules in that it does not stand on its own, but instead adds additional functionality to the Vision module. The reporting module adds dynamic reporting functionality to the Vision module, allowing you to display reports to Vision clients or to generate PDF files.

The reporting module offers flexible report generation, with a variety of components, charts and tables. Additionally, it supports the import of existing forms and images, allowing you to migrate from paper based tracking systems to an electronic system.

2.3.6 Mobile Module

The Mobile Module adds the ability to launch Vision Module projects on modern smartphones. This lets you keep track of your control system while moving around your facility. The Mobile Module can be combined with remote-access networking architecture to allow global on-the-go access to your control system.

© 2011 Inductive Automation

2.3.7 OPC-COM Module

The OPC-COM module gives Ignition the ability to connect to legacy ("classic") COM based OPC-DA servers. It supports OPC-DA 2.0 and 3.0.

Connecting to classic OPC servers

With the OPC-COM module installed, there will be a new option for COM based OPC servers when creating a server connection in the gateway. The OPC-COM module is primarily intended for use with local OPC servers, although it also provides basic support for remote connections.

Even when connecting locally, the application may run into the traditional difficulties of connecting to OPC servers. DCOM security settings on the machine can interfere with connections, and the OPC Core Components package must be properly installed before connections can be established.

Using data from classic OPC servers

After a connection to a server has been defined, the server will appear along side of other OPC servers (both COM and UA based) in the OPC Tag Browser. You can use these tags like any other ones - bring them into SQLTags, use them in Transaction Groups, etc.

2.3.8 Other Modules

The pluggable module architecture allows quick integration of new modules into the Ignition platform. From time to time new modules will be release which add additional features.

Driver modules

Drivers for the OPC-UA module are deployed as modules themselves. While they don't add a visible element to the system, they are loaded and upgraded in the same manner as other Ignition modules.

ActiveX Module

There is a free module available for separate download from our website that adds an ActiveX palette to the Vision module. This lets you use ActiveX controls in your windows. This module comes with some caveats, however. ActiveX doesn't play all that gracefully with Ignition, because it is written in Java. ActiveX controls will only work on Windows. They also draw themselves "on top of" the entire Vision client application. This means that nothing can overlap them, not even other windows or dropdown menus. Because of these technical limitations, this module is provided as-is, with limited technical support. These details aside, the ActiveX component can be a great way to integrate a full-fledged PDF viewer or web-browser into your Ignition Vision application.

2.4 Basic Usage

2.4.1 Gateway Navigation

Accessing the Gateway

The Ignition Gateway is accessed via a web browser. The web browser can be running on any machine that has network access to the host that is running the Ignition Gateway. By default, Ignition installs using **port** `8088`.

© 2011 Inductive Automation

Example

If the host's IP address was 10.0.28.30, you would access the Ignition Gateway via the URL:

```
http://10.0.28.30:8088
```

Gateway Sections

Across the top of the Ignition gateway you'll find several buttons that lead to the key sections of the server.

Home

The homepage shows a quick overview of the primary modules installed. From here you can:

- Launch Vision project clients.
- View the current status of the SQL Bridge module, and how many transaction groups are running.
- View the state of your device connections under the OPC-UA module.

Status

The status portal provides in depth information about various parts of the Ignition system. There are sub-pages containing information about:

- The state of the installed modules
- The current cluster map
- The status of all DB connections, OPC Server connections, and SQLTag providers.
- The status of the store and forward engine, including performance metrics and data cache information.
- Current designer and client sessions connected to the gateway.

Configure

This section is where all gateway/platform configuration is performed. See Gateway Configuration for more complete details.

In general, this is where you'll go to:

- Create new projects
- Create database connections
- Create connections to OPC servers
- Tune performance settings
- Modify SQLTags Historian data settings
- Manage users and roles and much more.

Launch Designer

This button directly launches the Ignition Designer.

2.4.2 Gateway Control Utility

The Gateway Control Utility, sometimes referred to as simply the "GCU", is a lightweight stand-alone application that can provide information about the Ignition gateway. It also provides basic administrative controls, such as stopping and restarting the server, and setting the ports used between the gateway

© 2011 Inductive Automation

and clients.

The Gateway Control Utility must run on the same machine as the Ignition gateway. It can be launched from the start menu under `Programs > Inductive Automation > Ignition > Launch Gateway Control Utility`.

There are normally two primary sections in the list on the left-hand side of the screen: the web server's status, and the *context* status.

Web Server Status

The "web server" refers to the Ignition server, and the embedded web server that it runs. Clicking on this section shows the overall run status of the server. It also provides several options:

Go to webpage
Launches a web browser to the gateway home page.
Thread dump
Downloads a file with the current states of all threads in the server, used by Inductive Automation for troubleshooting problems.
Port
Sets the primary, non-encrypted port used by clients to communicate with the server.
SSL Port
Sets the port that will be used by clients for SSL communication.

Context Status

The context status shows the state of the running *contexts.* Usually there is only one context, called "main". A context is a sand-boxed copy of Ignition running under the web server.

The context status page allows you to perform basic maintenance tasks:

Restart
Performs a "soft" restart on the context, in which only the context is restarted- not the entire server.
Upgrade
Allows you to upgrade the version of the context using a context package.
Reset Password
Allows you to reset the root password of the system. This is not normally considered a security risk, because the GCU can only be used from the machine the software is installed on, which should be secure. However, it is important to know that this function is here, so that the GCU can be removed if the machine can't be properly secured (for example, when the server is also used as a client).

2.4.3 Web Launching

Web-launching is the mechanism by which clients and designers are opened on a machine. They are *launched* from the Ignition gateway, download and run without requiring any installation steps.

How Web-Launching Works

Web-launching relies on *Java WebStart* technology. When the user clicks on a project or designer link in the Ignition gateway (or embedded in a separate website), they download a small *JNLP* file that describes the application. When this file is opened (usually automatically), Java is invoked on the user's machine and directed to the remote application. The application is downloaded and cached, and then executed.

© 2011 Inductive Automation

The running application (an Ignition client or designer) communicates with the gateway via HTTP. It is cached for increased subsequent launch speed, and can optionally install a link in the Start menu and on the desktop for easy access.

Troubleshooting Web Launch Problems

There are a few common problems that can cause difficulties with web-launching. Fortunately, they are often easy to fix.

No Java Installed
Web-launching clients and designers requires having Java version 5 or greater installed on the client machine. The Ignition gateway will detect whether Java is installed and offer help, but users are free to download and install Java on their own. Java can be installed by visiting http://www.java.com

No Network Connection
Web-launched clients depend on network connectivity to connect to the server. If the network is unavailable, the client cannot be launched. This is often a problem with clients and designers launched directly from desktop/start menu links.

Cached References to Modified Servers
The cached projects/launch shortcuts contain the address of the gateway machine. If the server is relocated, these links will no longer work. They can be updated by re-launching from the gateway.

2.4.4 Launching Clients

Clients are launched by going to the gateway homepage. See Gateway Navigation for more information about accessing the gateway.

There are three ways to run clients: Windowed, Full screen, and Applet. The mode can be chosen from the drop down next to the project name. Clicking on the project name will launch the project in the default mode. Certain modes may not be available, depending on project settings.

Windowed

The "Windowed" mode is the standard launch method. The client will be web-launched using Java Web-Start and will have its own window. In this mode, it will run as a full, independent application. After being launched, the browser can be closed and the project can be launched from a shortcut on the desktop.

Full Screen

The "full screen" launch mode is similar to the Windowed mode, and will also use web-launching to run the client as a full, independent application. In this mode, however, the client will occupy the full screen, and will not have a title bar. This mode is ideal for touch-screen display panels, and other displays where the Ignition project will be the sole focus on the screen.

Applet

Selecting "applet" launch mode will run the client application as an applet embedded in your web browser. Applet mode is most commonly used to integrate Vision projects into existing web sites, such as in a corporate intranet setting.

© 2011 Inductive Automation

Mobile

If you have the Mobile Module installed, you can launch projects on your smartphone or tablet as well. All the user has to do to launch a mobile client is to connect their mobile device to the wireless network and point the web-browser to the Gateway's LAN address. At this point, they'll be presented with a mobile-optimized version of the Ignition Gateway homepage, where they can select a project to launch. Note that projects must have at least one window defined and be enabled for mobile launching in order to show up in this list. After selecting a project and logging in, they can use the project like a normal project. To access the mobile project context menu, press and hold on your touch-sensitive device. A circular menu will appear allowing you to switch between pointer and pan/zoom mode, as well as options for logging out and entering text input.

2.4.5 Launching the Designer

The Designer can be launched from the Gateway homepage by clicking on the "Launch Designer" button. See Gateway Navigation for more information about accessing the Gateway.

It is possible to have multiple Designers open concurrently. Each Designer will lock the resources that it's modified, and other Designers won't be able to access them until they've been saved.

© 2011 Inductive Automation

Gateway Configuration

Part III

inductive automation
800.266.7798
www.inductiveautomation.com

3 Gateway Configuration

3.1 Gateway Configuration Overview

The gateway is the central location where all general services are configured in Ignition. Additionally, the gateway configuration section is where operations such as backing up the system, restoring, and managing projects are performed.

The gateway configuration settings cover the following broad categories:

System Management - Licensing, Backup/Restore
Module Management
Database Connectivity
OPC Connectivity
SQLTags
Security
Alerting

Other categories may also be available, depending on the modules installed.

3.2 Logging into the configuration page

Clicking on the `Configure` section of the title bar, or any link on the homepage that would take you to the configuration section, will bring you to a gateway log-in page.

Gateway access is controlled by an authentication profile, in the same way that projects and the designer are protected. The gateway settings dictate which roles are allowed to have access. It is important that the gateway be kept secure, therefore you should quickly change the default authentication settings.

Default Login

When the system is first installed, the gateway can be access with the following credentials:

```
Username: admin
Password: password
```

As mentioned above, it is strongly suggested that you quickly change these default settings to something more secure. See the Managing Users section for more information.

3.3 Basics

3.3.1 Basic Gateway Settings

The basic gateway settings are found under `Configuration > Gateway Settings`. They define high-level settings that apply to the entire gateway.

System Name
A unique name for this Ignition installation. It is used to distinguish between this server and others on the network when working with multiple Ignition installations.

System Authentication Profile
The authentication profile used to secure access to the Gateway, as well as to the Designer.

Gateway Config Roles

© 2011 Inductive Automation

A comma-separated list of roles, one of which will be required in order to log into the Gateway's configuration section. These roles roles should be defined in the System Authentication Profile.

Status Page Roles
Required roles to access the Gateway's status section. Leave blank to remove security restrictions for this section.

Home Page Roles
Required roles to access the Gateway's home section. Leave blank to remove security restrictions for this section. Note that this is only used to limit access to the homepage itself, each project will have its own authentication profile for limiting access to the runtimes.

Designer Roles
The roles that will be granted access for logging into the Designer.

Use SSL
Forces the clients to use SSL encrypted communication when talking to the gateway. It is recommended that you purchase and install a genuine SSL certificate if you use this option. See the guide in the Deployment section of this manual.

Persist Alerts
Whether or not alert properties such as acknowledgment should be persisted across Gateway restarts.

Launch Link Script Policy
Controls how the HTML that launches Clients and Designer functions. If set to `JavaScript`, the links will use javascript to attempt to launch directly using the Java browser plugin. If set to `Direct`, the links will be direct links to the `*.jnlp` files that launch the Client or Designer.

Allowed JREs
Which versions of the Java Runtime Environment will be allowed to web-launch clients and designers.

Designer Memory
The maximum memory that the designer may use.

Disable Direct3D / Disable DirectDraw
These advanced properties affect launched Clients and Designers on Windows OS only. These flags control whether or not the Java Swing windowing subsystem may use Direct3D and/or DirectDraw. Disabling these features may incur a performance penalty, but might be required for some video cards that have faulty DirectX drivers.

Scheduled Backups
These 4 properties (enable, backup folder, backup times, and retention count) control the Gateway's scheduled backup system. This system is capable of automatically making a Gateway backup and storing it to a folder path, which may be a network path. When you enable this system, you must specify a destination folder. This may be a local folder, for example `"C:\backups"` or `"/var/backups"` or a network path such as `"\\fileserver\backups"`.

The scheduled backup system works on a schedule that is specified using UNIX crontab syntax. This is a standard format for specifying a basic schedule. The format consists of five space-separated fields, one for minute, hour, day-of-month, month, and day-of-week. The special character * means "all". Slashes can be used to indicate that values should be stepped, for example, */5 in the minutes field means "every 5 minutes", or 0:00, 0;05, 0:10, etc. Some examples:

`5 * * * *` Once an hour, on the :05 minute. 0:05, 1:05, 2:05, etc.
`*/15 * * * *` Every 15 minutes, on the quarter-hour. 0:15, 0:30, 0:45; 1:00, 1:15, etc.
`30 5 * * Mon` Every Monday at 5:30am

© 2011 Inductive Automation

`* 6-14 * * *`	Every minute, but only between 6am and 2pm
*/5 8-17 * * 1-5	Every 5 minutes between 8am and 5pm but only during the week (1-5). 0=Sunday, 1=Monday, etc.
0 1 5 * *	Once a month, on the 5th day at 1am

If something is wrong with the scheduled backup system it will store error messages to the Gateway logs.

3.3.2 Gateway Homepage Customization

It is possible to modify the options available on the gateway homepage from the Homepage Config section, a tab under `Configuration > Gateway Settings`.

This section allows you to specify which panels are shown, whether they are initially expanded or collapsed, and the order in which they appear.

3.3.3 Setting the Port

By default, the Ignition server runs on port `8088`. There are a variety of reasons why it might be necessary to change this, such as the port already being used by another application, or the desire to run on a more "user-friendly" port, such as 80.

The port can only be set through the Gateway Control Utility. It cannot be changed from the gateway configuration portal.

If the port is already in use, the Ignition gateway will not be allowed to start. In this case, the Gateway Control Utility may be used from the server machine to select a different port for the server.

3.3.4 Resetting the trial period

When unlicensed, the Ignition gateway will run for two hours at a time. After the trial period has expired, all unlicensed modules will be stopped.

It is possible to reset the trial period by logging into the gateway configuration section and selecting "Reset" from the trial banner. It is possible to restart the trial period any number of times. Depending on the module, additional action may need to be taken. For example, the Vision Clients require the user to log out and back in in order to continue the trial.

3.3.5 Activation

3.3.5.1 Online Activation

All activation activity is performed in the gateway configuration portal under `System > Licensing`. The general topic of activation is described in the introduction under Licensing, Activation, and Trial Mode.

If you have been issued a CD Key and wish to activate online:
1. Click on the "Purchase or activate..." link on the licensing page.
2. Click on "Activate"
3. Enter your CD Key
4. The request will be processed over the internet. If a connection is not available, you will be redirected to a page that allows you to perform an offline activation.

© 2011 Inductive Automation

3.3.5.2 Offline Activation

Offline activation is used to activate servers when an internet connection isn't available. The process consists of generating a secure file, transferring it to Inductive Automation, and receiving back a corresponding license file.

To activate off-line, follow the same steps as outlined in the Online Activation section. After entering your CD Key, you will be presented with a screen where you can download your activation request file.

Methods of Transferring the Activation Request

The activation request file may be used on the Inductive Automation website to generate an activation file instantly. Additionally, you may email it to support@inductiveautomation.com. It will be processed and returned within 1 business day.

Installing the License File

Once a license file has been generated from an activation request, it can be loaded by returning to the Licensing section of the gateway configuration portal.

3.3.5.3 Unactivation

Only one Ignition gateway instance is allowed to be activated at a given time, for a given CD Key. If you would like to activate Ignition on a different server, you must first *unactivate* the previous server.

To unactivate, go to `System > Licensing`. On that page you will see the currently installed license, with the option to "unactivate" at the bottom of the display. Clicking this link and following the instructions will initiate the unactivation procedure.

Unactivation is virtually the exact opposite of Activation. In the process, an "unactivation request" will be generated. The software will be unactivated immediately, but a new activation will not be available until the unactivation request is received by Inductive Automation. There are both online and offline options for transferring the request, as with activation.

3.3.5.4 Updating the License

If you wish to modify your license in order to add or remove modules, or change the level of a particular license, you will need to contact Inductive Automation. The modules will be adjusted for the CD Key, and you will then be able to re-activate the system using the same key. Once your new license file is installed, the Ignition server will be updated with the desired module licenses.

3.3.6 Gateway Console

The Gateway Console provides a wealth of information about the running state of the gateway. It is located under `System > Console`, in the gateway configuration portal. Most of the features in this section are for advanced troubleshooting, and are not often consulted in normal operation. Of all of the tabs in this section, the Log Viewer is the most useful for system administrators.

Log Viewer

The log viewer shows the most recent output of gateway "loggers"- units in the gateway application that output information.

© 2011 Inductive Automation

Levels

The Levels tab shows all of the registered system loggers, and the level of detail that they should record.

Threads

This section shows the current state of all system threads.

Execution Stats

Show a list of all registered "executors"- tasks that perform repeat operations.

Cluster

Provides a console where test and advanced debug messages may be sent to the cluster.

3.4 Projects

3.4.1 What is a Project?

An Ignition *project* is a unit of configuration that consists of:
- Windows
- Transaction Groups
- General Settings
- Security Settings

Each runtime client or designer can operate on one project at a time. If a project contains viewable elements (windows, reports) a launch link for it will appear on the gateway homepage. Otherwise, the project will run in the gateway and will not have a client runtime.

There is no limit to the number of projects that can be created on a gateway.

What is *not* part of the project?

SQLTags providers, database connections, OPC server connections and OPC-UA module device configurations are all *not* contained in a project. These settings are shared by all projects on the system.

3.4.2 Project Management

Project management is performed under `Configuration > Projects` in the gateway. Some project management can also be performed through the designer. See Designer Project Properties for more information.

Creating a new Project

To create a new project, click on "Create new project" from the project management page. To create a new project you'll define:
- Name - A unique name for the project in the system. Note: it is not advisable to change this after it's been created, instead, change the *Title* if you want to change how the project appears later.
- Description - Purely for informational purposes for the user.
- Title - How the project will appear to users.

© 2011 Inductive Automation

Additionally, there are a few crucial properties that dictate how the project is accessed and how elements inside of it act:

Authentication Profile
The profile to use for granting access to the project. For more information, see the Security section.

Default Database
All elements of the project will use this database connection unless explicitly specified otherwise.

Default SQLTags Provider
The primary SQLTags provider for the project. Most installations will likely only have one provider, but in situations where there are more than one, this is the provider that will be used by default.

Deleting Projects

Projects can be deleted by selecting the "Delete" option to the right of the project name in the list. Be aware that this action cannot be undone, and once a project is deleted it is gone forever (unless it can be recovered from a backup or auto-backup. See the Backups section for more information).

Copying Projects

Projects can be cloned easily using the "Copy" link next to the project's entry. This is useful for creating a "snapshot" of a project before starting major changes, or for creating a starting point for a new project based on an old one.

3.4.3 Project Versioning

Each project can have two distinct versions at once: the *Staging* version and the *Published* version. By default, a new project is configured to be in *Auto* publish mode, which means that the two versions are always identical. However, if you change a project to be in *Manual* publish mode, then you can explicitly publish a project in the Designer.

Published vs Staging

The general idea between having a published version and a staging version is to allow you to save a project, and then test out the changes before "publishing" those changes to a production environment. Under normal conditions, Vision module clients run the published version of a project. However, by launching a client in a special mode (from the Designer or from the Config section of the Gateway), you can launch a client that runs the staging version of that project. This staging client will receive updates on every save, where the production clients receive updates only on publish. This lets you test out your changes to the project in an actual client, which is more realistic than the Designer's preview mode.

Not all aspects that comprise a project use this system. It is primarily intended for systems such as the Vision module's clients. Features that run persistently on the Gateway, such as SQLTags, the SQL Bridge's Transaction Groups, and Gateway-side scripting always run the most recently saved changes (the Staging version). Since these features by definition must run in exactly one place, they cannot be effectively "tested out" by simultaneously running a staging version alongside a published version.

Project Versioning and History

Each project keeps a log of recent changes. These include both saves and publishes. Every save increments a number called the "edit count" for the project, which can be used like a serial number. The

© 2011 Inductive Automation

user, time, affected resources, and a commit message (see below) are logged as well.

Commit Messages

A project may be configured to prompt the user making changes to describe those changes, either on every publish event, or on every save and publish event. These messages, called *commit messages*, are used to describe the changes that have been made to the project. By inspecting the project's history and reading these commit messages, you can learn what changes have been made to the project, for what reason, and by whom.

See also:
Project General Properties

3.4.4 Importing and Exporting Projects

There are two primary ways to backup and restore projects.

System backup / restore

A project can be backed up as part of a full system backup. A system backup includes all of the projects in the system, in addition to all of the settings. Restoring a system backup will replace all current projects and settings on a gateway. See Backups for more information.

Project export / import

Projects can exported and imported individually through the project management page. Click the *download* link to retrieve the *.proj file for the project. To restore, click the *upload project* link below the project table.

Project exports only include project-specific elements, like windows and groups. They do not include gateway settings, like database connections and device configurations.

3.5 Modules

3.5.1 Module Management

All module configuration is performed under `Configuration > Modules`. Note that this section is used solely for adding, removing, and restarting modules. Modules integrate their settings into the gateway configuration tree, and therefore do not offer settings in this section.

Installing, Upgrading and Uninstalling Modules

Modules can be installed by selecting *Install or Upgrade a Module* below the module list. To uninstall a module, click *uninstall* next to its entry in the table.

Modules are hot-swappable, so these actions can be performed while the system is running. Furthermore, the isolated nature of modules ensures that performing one of these actions will only affect that particular module, and any modules which depend on it. For example, uninstalling the SQL Bridge module will not affect any running Vision module clients.

© 2011 Inductive Automation

Restarting a Module

Modules can be restarted by clicking the *restart* button next to their entries. As mentioned above, the isolated nature of modules means that the other modules will not be affected by the restart (unless they depend on that particular module).

Module Status

The installed module list also provides some basic information about the state of the module. The version, license and state are all displayed in the list. Module licensing is performed centrally in `System > Licensing`, so the values here are only for information purposes.

3.6 Databases

3.6.1 Databases Overview

Database access is at the heart of the Ignition platform, enabling you to create robust data-centric systems. Relational "SQL"-based databases are extremely common in modern companies, and offer a tremendous amount of power and flexibility in storing, calculating, and manipulating data. By connecting Ignition to one or more databases, you can leverage this power to create systems that expose data, store historical information, and more.

Uses of Databases in Ignition

The following are a few places that databases are used in Ignition. While connecting to a database is not required for basic status and control functionality, it can dramatically increase the possibilities that the system offers.

Historical Data Logging
Logging data for historical analysis, either through SQLTags Historian or with the SQL Bridge module, requires a database connection. Databases are great at handling historical data, and by using a standard relational database your data is stored in an open format that can be used in many ways.

Reports, Graphs and Charts
The Vision module makes it easy to present data stored in databases in a variety of ways. You can quickly create charts that show performance over time, locate anomalies, detect trends, and more. Furthermore, it's important to remember that it is possible to pull data from any database that Ignition is connected to, even if the data wasn't placed there by Ignition. This means you can tie in data from other sources or areas of your company, such as pulling in inventory and staff information, as well.

Storing Alarm Logs
Store alarm information historically and examine it later for patterns or trouble spots.

Database-driven SQLTags
It is possible to use a SQL database as your SQLTags repository. Any other Ignition system with access to the database will be able to share and contribute tags, allowing you to create highly integrated distributed systems. For example, multiple plant sites could use SQLTags to report current status over a secure network connection to a central corporate headquarters.

Getting Started with Databases

The first step in using a database with Ignition is to identify or setup a database server. Many companies

© 2011 Inductive Automation

already have database servers maintained by their IT departments. If you do not, or wish to set up your own database server for Ignition, the Supported Databases section offers some advice on choosing a database vendor.

Once you've identified a server, all you need to do is create a connection to that server to get up and running.

3.6.2 Supported Databases

Ignition has been tested with the following databases, and can connect to them directly after installation. It is possible to connect to other databases by installing additional JDBC drivers (the Java database connection specification), which are often provided by database vendors.

Full support

Database	Version
MySQL	5.0+ for full support. Ignition will connect to 4.x, but many features such as SQLTags have not been tested.
Microsoft SQL Server	2005, 2008, full and express editions. Ignition will connect to 2000, but has not been fully tested.
Oracle	10g, 11g, full and express.
PostgreSQL	8.0+

Limited support

Database	Version
Microsoft Access	Access support is very limited, and should only be used to integrate existing data into the project, not for storing new data.
Other JDBC drivers	Due to variances in databases, some features may not work fully through other non-tested JDBC drivers. However, it is usually possible to get full functionality though the careful use of the *database translator* feature.

Choosing a database

If you are new to working with SQL databases and are trying to choose a vendor, there are several factors to weigh:

Existing company usage - Many companies already use SQL databases for other purposes, and thus most IT departments already have a defined standard. Going along with your company's existing standard is usually recommended, as there will already be staff available who are knowledgeable about the system. Furthermore, you may be able to tie into your company's existing database system instead of maintaining your own.

Price and Features - The fully supported databases above vary dramatically in price. Some systems can cost thousands of dollars, but may have a free "express" edition that will work perfectly well for your requirements. Others offer advanced features such as clustering, which are either not offered or difficult to configure in the other systems. It is therefore important to clearly define the features and capabilities that you need.

Most common among Inductive Automation users - choosing a database that is commonly used by Inductive Automation users means that you are more likely to find examples and help in the forum, among other benefits. The supported database list above is sorted according to our current user install base.

© 2011 Inductive Automation

3.6.3 Database Connections

3.6.3.1 Creating and Editing Connections

Database connections are managed in the gateway under `Databases > Connections`. To create a new connection, click the *Create new Database Connection* link below the connection table.

Select a driver

Select the appropriate database driver for the server that you'll be connecting to. If a suitable driver isn't present in the list, you may need to install a new JDBC driver.

Configure Connection

After selecting the driver, you'll configure the settings for the connection. Some settings, such as the *Connect URL* may be specific to the driver that you're using.

Connection Settings

Connect URL	A string that instructs the driver how to connect to the database. This string will include the server address, and may include the port, instance name, database name, etc. The format and parameters will depend on the driver being used.
Username	The username to use when connecting. Some databases support other authentication methods, such as Windows authentication, in which case this field would not be used.
Password	The password to use for the given username.
Failover Datasource	The connection to use when this connection is not available.
Failover Mode	How to handle failover and recovery. See below for more information on failover.

Advanced Settings

There are a variety of advanced settings that should not need to be changed under normal circumstances. Their descriptions are shown on the settings page.

Database Connection Failover

Database connections support "failover", by which objects that use that connection will use a different connection if it is unavailable. The *failover datasource* determines which connection will be used, and the *failover mode* determines when, if ever, the connection will switch back to the primary connection. *Standard* mode dictates that the secondary connection will be used only until the primary connection is available again. *Sticky*, instead, will continue to use the secondary connection until that connection fails, or until the system is restarted.

3.6.3.2 Monitoring Connection Status

Database state can be monitored from the Status section of the gateway, under `Status > Database Connections`.

The status panels show the current state and a fault message, if applicable, or throughput statistics if the connection is active. When a connection is not available, it will be re-tested every 10 seconds, and the status will be updated.

3.6.3.3 Connecting to Microsoft SQL Server

Selecting the Driver

To connect to SQL Server, use the **Microsoft SQLServer JDBC Driver**. This is the official Microsoft

© 2011 Inductive Automation

JDBC driver, and works with SQL Server 2000 (sp3) and above.

Selecting the Database

Unlike some other drivers, the name of the database that the connection will target is defined in the extra connection properties by the `databaseName` parameter.

Connecting to SQL Server 2000, or a server with a well-known port

To connect to a server using a well known port, use the following *Connect URL* style:

```
jdbc:sqlserver://hostname:port
```

Hostname can be an IP address or the server name, if local on the network. The port, by default, is **1433** .

Connecting to a SQL Server named instance

To connect to a named instance (standard in SQL Server 2005 and later), use the following *Connect URL* syntax:

```
jdbc:sqlserver://hostname\instancename
```

The default instance name for SQL Server Express is **SQLExpress**. Therefore, an common connection URL for connecting to the local database is:

```
jdbc:sqlserver://localhost\SQLExpress
```

Using Windows Authentication

To use Windows Authentication, where the connection is authenticated using the identity of the user running the Ignition gateway, you must first install the Microsoft JDBC driver package. The driver package can be found at: http://msdn.microsoft.com/en-us/sqlserver/aa937724.aspx

In the install directory of that package, you will find a *sqljdbc_auth.dll* file. Copy the correct file for your architecture into the following two directories:
`<installation directory>\Ignition\tomcat\webapps\ctx0\WEB-INF\lib`
and
`<Program Files directory>\Java\<jre folder>\bin`
`(ie: C:\Program Files\Java\jre6\bin)`
Then restart the Ignition service.

In your database connection, add the following Extra Connection Parameter:

```
integratedSecurity=true;
```

Common Problems

TCP/IP Communication Not Enabled
SQL Server requires that you explicitly turn on TCP connectivity. To do this, use the *SQL Server Configuration Manager*, located in the Start menu under "Microsoft SQL Server>Configuration Tools". Under "SQL Server Network Configuration", select your instance, and then enable TCP/IP in the panel to the right. You will need to restart the server for the change to take affect.

Window Firewall
When connecting remotely, make sure that Windows Firewall is disabled, or set up to allow the necessary ports. Normally ports **1434** and **1433** must be open for TCP traffic, but other ports may be required based on configuration.

SQL Server Browser Process Not Running
To connect to a named instance, the "SQL Server Browser" service must be running. It is

© 2011 Inductive Automation

occasionally disabled by default, so you should verify that the service is not only running, but set to start automatically on bootup. The service can be found in the Windows Service Manager (Control Panel>Administrative Tools>Services).

Mixed Mode Authentication Not Enabled

Unless selected during setup, "mixed mode" or "SQL authentication" is not enabled by default. This mode of authentication is the "username/password" scheme that most users are used to. When not enabled, SQL Server only allows connections using Windows Authentication. Due to the ease of using SQL Authentication over Windows Authentication, we recommend enabling this option and defining a user account for Ignition.

To enable this, open the SQL Server Management Studio and connect to the server. Right click on the instance and select "Properties". Under "Security", select "SQL Sever and Windows Authentication mode".

3.6.3.4 Connecting to MySQL

Selecting the driver

After creating a new connection from the `Databases>Connections` section of the gateway, select **MySQL ConnectorJ**.

Connect URL

MySQL uses the following URL format:

```
jdbc:mysql://hostaddress:3306/database
```

The *hostaddress* will be the address of the machine with MySQL installed, for example: *localhost, 192.168.1.1, db-server,* etc.
The *database* parameter will dictate which *database schema* the connection will target. It's important to understand that a MySQL server can host many *database files*. The connection will target one database.

Extra Connection Parameters

By default, there is one extra connection parameter defined, `zeroDateTimeBehavior`. It is usually not necessary to add more parameters.

3.6.4 Database Drivers

3.6.4.1 What is JDBC?

JDBC stands for the Java DataBase Connectivity API. It is a standardized way for Java-based applications to interact with a wide range of databases and data sources. A *JDBC Driver* enables Ignition to connect to, and use data from, a particular database system.

JDBC in Ignition

Ignition, being a Java based application, leverages JDBC in order to connect to a variety of data sources. This enables Ignition to offer a standardized set of functionality on a wide range of different systems and databases. This includes databases such as MySQL, Microsoft SQL Server, and Oracle, but additionally other lesser-known systems as well, provided the manufacturer offers a JDBC driver for the system.

© 2011 Inductive Automation

JDBC vs. ODBC

JDBC differs from ODBC (Microsoft's OpenDataBase Connectivity standard) primarily in the fact that JDBC is written in Java, and thus can be used without modification in cross-platform environments. Additionally, whereas ODBC is a complex standard that is becoming technically out-dated, JDBC is a modern, clean specification for cross-vendor database access.

3.6.4.2 Can I connect using ODBC?

While it is indeed possible to connect to an ODBC data source through the use of the JDBC-ODBC bridge, this is generally not advised. The bridge is designed to offer a minimal amount of functionality, and is considered a "transitional solution", meaning that it should only be used when JDBC is not available. In other words, if a JDBC option is available, ODBC should **not** be used.

Since most commercial databases offer JDBC drivers, transition is usually as simple as recreating your connections inside of Ignition. The lack of a JDBC connection inside of Ignition does not necessarily indicate that JDBC isn't available for your particular database. Licensing restrictions sometime prevent the inclusion of drivers with 3rd party software. Therefore, before using ODBC, due diligence should be taken to verify that no JDBC solution is available.

3.6.4.3 Adding a JDBC driver

To add a new JDBC driver to Ignition, click the *Create new JDBC Driver* link from `Databases > Drivers` page. In order to install a new driver, you'll need the Java JAR file that contains it and any other required JARs, as well as the full name of the driver class. This name is provided in the manufacturer's documentation.

Main Properties

Classname	The full name of the JDBC driver. Should be provided in the manufacturer's documentation.
JAR files	The core JAR file containing the driver, as well as any others required by it.

Driver Defaults and Instructions

These properties will be used as defaults when creating new connections against this driver.

Driver Type	The brand of database. This is used for optimizations in the gateway, if in doubt, select GENERIC
URL Format	A template/example of the jdbc connection string to use. The driver documentation should have examples of this string.
URL Instructions	Free form instructions that will be shown to help the user create a connection.
Default Connection Properties	Any additional properties to add by default to the connection string.
Connection Property Instructions	Tips about which connection properties might be useful.
Default Validation Query	The default query that will be used to verify that the connection is available.

SQL Language Compatibility

Default Translator	The database translator that will be used by default for connections from this driver.

© 2011 Inductive Automation

3.6.4.4 Database Translators

Despite the presence of a SQL standard, many database system vary in how they implement or accomplish various tasks. The JDBC driver system tries to hide these differences as much as possible, but unfortunately some differences persist.

The *database translator* system in Ignition navigates these differences as they apply to the system. It provides a way to define certain key operations that are commonly different between database vendors, such as creating auto-incrementing index columns, and the keywords used for different data types.

Translator Management

Database translators are managed in the gateway under `Databases > Drivers > Translators (tab)`. Ignition comes pre-configured with translators for the major supported databases, but it is possible to edit and remove them, as well as create new translators. It should only be necessary to create a new translator when adding a new JDBC driver for a database that does not share syntax with any of the existing translators.

Creating a New Translator

Each field of the database translator will define a pattern that will be used, with special token markers to indicate places where other values will be placed. For example, the default *Create Table* entry looks as follows:

```
CREATE TABLE {tablename} ({creationdef}{primarykeydef})
```

In this example, *tablename*, *creationdef*, and *primarykeydef* are all tokens that will be expanded. The token *tablename* will be replaced directly with the table, but *creationdef* will be a list of columns, and *primarykeydef* will be the phrase created by the *Primary Key Syntax* entry in the translator.

Many tokens only apply to certain entries. The possible tokens are as follows:

Token	Description
tablename	The name of the table being created.
indexname	The name of the index to create, when adding a column index to the table.
primarykeydef	A clause that will define a primary key for a new table.
creationdef	The list of columns to create in the table
alterdef	A list of columns to add/remove/modify in the table
columnname	The name of a column
type	The data type of a column
limit	The value of the limit clause

Other Translator Properties

Limit Position

Defines where the limit clause should be placed. With *Back*, the limit will be placed at the end of the query. *Front* will place it directly after the "SELECT" keyword.

Column Quote Character

All columns will be created and accessed with the defined quote, which tells the database to use a specific casing, as well as avoiding collisions between the column name and database keywords.

Supports Returning Auto-Generated Keys / Fetch Key Query

Indicates whether the JDBC driver supports the return of generated keys. If the driver does not support this feature, the *Fetch Key Query* will be used to retrieve the last key.

© 2011 Inductive Automation

Date Types
The keywords that will be used when creating columns of the given types.

3.7 Store and Forward

3.7.1 Store and Forward Overview

The store and forward system provides a reliable way for Ignition to store historical data to the database. Systems such as SQLTags Historian, Transaction Groups, etc. use the store and forward system in order to ensure that data reaches its destination in the database, and is stored in an efficient manner. The store and forward system can be configured in a number of ways, offering both memory buffering for performance and local disk caching for safe storage.

Primary Features and Benefits

The store and forward system offers a number of benefits over other systems that log directly to the database:

Data loss prevention - Data is only removed from the system when the write to the database has executed successfully.
Guaranteed Ordering - The store and forward system ensures that data is forwarded in the order that it arrived.
Enhanced performance - By first buffering the data in memory, the store and forward system can optimize writes, and prevent the originating systems from blocking. This means that the system is less likely to lose data samples in the event of system slow downs.

Store and Forward Data Flow

Although the system offers settings that can affect the pipeline, by default the data flow occurs as follows:

1. Data is generated in some system
2. Data is placed in a memory buffer
3. If not removed from memory buffer in some time (the Write Time), or if a certain amount of data accumulates (Write Size), is placed in the local cache.
4. The data sink, based on a database connection, pulls data in order from the local store, and then the memory buffer, based on the "Write Time" and "Write Size" settings under "Forward Settings".
5. If the data fails to forward, either due to an error in the connection or in the data itself, it is returned to the buffer or cache.
6. If the data errors out too many times, it becomes *quarantined.*
7. Quarantined data can be managed through the gateway, and can be deleted or un-quarantined, if the error has been resolved.

Understanding the Forward Triggers
Data is forwarded from one stage to the next based on the "Write Time" and "Write Size" triggers. These settings work as an "either/or" manner, meaning that if either of them is surpassed, the data will be forwarded. One important point to note is that *the Write Size setting influences the transaction size* of similar data to be forwarded, and therefore can have a big impact on performance. As a result, the Write Time should normally be used as the controlling factor, with the Write Size set to something that will provide reasonable transactions, like 100.

3.7.2 Engine Configuration

Configuration of the store and forward engines is performed in the gateway under `Databases > Store`

© 2011 Inductive Automation

and Forward. Store and forward engines are directly correlated to database connections, and are automatically managed so that each connection has an engine defined.

Tip: Create multiple database connections pointing to the same database if you wish to configure multiple store and forward engines for different purposes.

Store and Forward Settings

The settings of a store and forward engine define how and when data is moved through the system. It is advisable to understand these settings and set them carefully in accordance with your goals.

Memory Buffer Settings

Memory Buffer Size
The number of records that can be stored in the memory buffer, the first stage of the store and forward chain. Other settings define when the data will move from the memory buffer forward, this setting only determines the maximum size. If the max size is reached, additional data will error out and be discarded. The memory buffer cannot quarantine data, so if there are errors and the disk cache is not enabled, the data will be lost.

If set to 0, the memory buffer will not be used.

Store Settings

These settings apply to the local disk storage cache.

Enable Disk Cache
Turn on the hard-disk cache. Data will be stored here if it cannot be forwarded in a timely manner. The cache also stores quarantined data (data with errors).

Max Records
The maximum size of the cache. After the max is reached, data will back up into the memory buffer, and once that is full, dropped.

Write Size
The number of records that should be accumulated in the memory store before written to the cache. Writing data in blocks can increase performance, but too large of a size increases the risk of data being lost in the event of a power outage or system failure.

Write Time
The max age of records in the memory buffer before they are stored to the cache. This setting is used in combination with the write size in order to give the forwarder the opportunity to retrieve data directly from the memory store and avoid the write to disk entirely.

Forward Settings

These settings govern when data will be forwarded to the database. The data will be pulled first from the local cache, and then from the memory store. When no data is present in the cache, it is pulled directly from the memory store.

Write Size
Same as disk cache setting above.

Write Time
Same as disk cache setting above.

© 2011 Inductive Automation

3.7.3 Store and Forward for Reliability

The store and forward system settings, while seemingly limited, offer a good deal of flexibility in tuning. Different types of situations and goals will likely require different configurations.

When the safety of the data is a concern, the goal is to get the data stored to disk as quickly as possible in order to minimize risk of loss due to a power outage or system failure. The local cache plays a crucial role in this, allowing the system to store data locally for any amount of time until the remote database can accept it. This protects against network failures and database failures, as well.

By setting the *write size* and *write time* of both the local cache and forwarder to low values, the data will spend less time in the memory buffer. While the memory buffer can be set to 0 in order to bypass it completely, this is not usually recommended, as the buffer is used to create a loose coupling between the history system and other parts of Ignition that report history. This disconnect improves performance and protects against temporary system slowdowns. In fact, it is recommended that for reliable logging this value be set to a high value, in order to allow the maximum possible amount of data to enter the system in the case of a storage slowdown.

Recommended Settings

These settings are merely a starting point, and should be adjusted to fit your goals.

Memory Buffer Size
1000 or higher. While the data won't reside in here for long, a high value will allow the data to enter the store and forward system, as opposed to being lost if the maximum is hit.

Disk Store - Enabled

Max Records
500,000 or higher. Like the memory store, if the maximum is reached data will be lost, so it is best to set the value high to protect against long periods of time without database connectivity.

Store Settings - Write Size
Very low, in order to get data into the cache as quickly as possible. Moving from the memory buffer to the disk store does not use transactions as much as forwarding to the database, so a low value should not impact performance too dramatically. A value of 1 is possible, though that would force all data to go to the cache before going on to the database. A value of 10 would likely be a good starting point.

Store Settings - Write Time
This should be the controlling factor in trying to get the system to forward as quickly as possible, minimizing the time that data in the memory buffer. If the write size is 1, this setting will be of little consequence, but if the value is greater than one, careful consideration should be given to this value. Ultimately, this value should only be as large as what you would be willing to lose if there were a power failure.

Forward Settings - Write Size
This value should be set to a decent size to increase transaction throughput. 100 is a good value.

Forward Settings - Write Time
This setting should be less than the Store Write Time, in order to avoid writing to the store when the target database is available.

© 2011 Inductive Automation

3.7.4 Store and Forward for high-speed buffering

When configuring the store and forward system for high-speed buffering, you are expecting the case that data will come in quick bursts. By buffering the data, the system can accommodate more information than would be possible going directly against the database.

The key points in configuring a buffering system is to avoid expensive operations like storing and reading from the local cache, and to set the memory buffer large enough to accommodate the expected burst sizes.

Recommended Settings

Memory Buffer
500 or higher. It should be high enough to accommodate several bursts of data. For example, if you expect data to be logged at 100 ms burst for 10 seconds at a time, 100 records would be the minimum value. Data will be forwarded as it comes in, according to the forward settings, but you should not rely on any particular throughput in order to avoid data loss.

Disk Store - Disabled
Depending on your requirements, the disk store should be disabled, or at least set to have high write size/count settings. Writing and reading from the cache is much slower than memory, so it is desirable to avoid it. Of course, the cache should only be disabled if it is ok to lose some data, should the database connection be down for a period of time.

Forward Settings - Write size
Should be larger than the expected burst size. Burst data will be from the same source, and therefore will benefit heavily from the optimizations in the buffer.

Forward Settings - Write Time
Should be balanced in order to give the buffer time to received multiple records that can be optimized, as describe in Write Size above. However, it should not be so long that too much data becomes scheduled to write, which could cause a system slowdown/back up.

3.7.5 Engine Status Monitoring

The store and forward engines can be monitored under the Status section of the gateway, under the `Store & Forward` menu. Each store and forward engine will be listed, each displaying the current throughput and capacity of its memory buffer and local cache.

Statistics

Availability
Shows the status of the engine, each store, and the database.

Pending
The number of records waiting to be forwarded in that section.

Quarantined
The number of quarantined records for the cache.

Store and Forward Statistics
Shows the throughput, number of transactions, and duration statistics. It is important to remember

© 2011 Inductive Automation

how the data flows when interpreting the statistics. The number of rows that have gone to the database will be the number forwarded from the local cache, and then the number forwarded from the memory buffer, minus those that entered the cache from there.

3.7.6 Data Quarantining

Quarantined data is data that has errored out multiple times during attempts to forward it. It has been removed from the forward queue in order to allow other data to pass. The most common reason for data quarantining is an invalid schema in the database for the data that is being stored.

Quarantined data will be held indefinitely until it is either deleted or re-inserted into the queue manually.

Quarantined data is controlled from the *Quarantine Control* tab under `Databases > Store and Forward`. The data is listed according to store and forward engine and the data format, with a description, the error that caused the quarantine, and the number of quarantined records. Next to the record, there are options to *Delete* and *Retry*.

Delete
Permanently delete the data in the selected row. All transactions of the selected type will be deleted.

Retry
Un-quarantine the data and place it back in the forward queue.

3.8 OPC

3.8.1 What is OPC?

OPC is a specification for the transport and use of industrial data. It is published and maintained by the OPC Foundation, an organization comprised of hundreds of member companies that strives to ensure interoperability on the plant floor and beyond.

History

The OPC-UA specification is the latest specification in a line spanning back to the mid '90s. The original OPC specifications used Microsoft DCOM technology to provide a uniform way for industrial applications to share data. There were several separate specifications that provided functions such as Data Access (OPC-DA), Alarms and Events (A&E), Historical data (HDA) and more.
DCOM always proved difficult to work with, and by 2004 it was clear that a more modern solution was needed. Therefore, a new specification was developed that used common networking principals instead of DCOM, was platform independent, and combined the various separate specifications into one: OPC-UA.

Clients and Servers

When discussing OPC (as the specifications are often called collectively), it is common to hear about "OPC servers" and "OPC clients". An OPC Server is a piece of software that implements the OPC interface and provides data. An OPC Client is an application which connects to an OPC Server and uses the specification to retrieve and work with data.

The Ignition platform inherently offers OPC-UA client functionality. That is, even with no modules installed, the gateway can connect to any compliant OPC-UA server and work with data. With the addition of the OPC-UA Module, Ignition becomes an OPC server as well, hosting device drivers that read and publish data.

© 2011 Inductive Automation

The OPC-COM module is available to provide client access to older, DCOM based, OPC-DA servers.

Technology

The OPC-UA specification offers a wide range of flexibility in choosing technologies, from the transport mechanism, to the way data is encoded, to the encryption used to secure the data.

Ignition supports the UA/TCP transport with the UA/Binary encoding scheme for maximum performance. Additionally, Ignition supports all of the common encryption schemes.

This means that Ignition connects to OPC-UA servers (and allows connections from clients) over TCP/IP, using encryption, and sends data by first encoding it into an efficient format defined by the OPC-UA specification. This is in contrast to other schemes outlined in the specification, which can use web services and XML encoding, and are not as efficient.

3.8.2 OPC Connections

3.8.2.1 Connecting to OPC-UA

OPC-UA Connection

An OPC-UA Connection is used to communicate with an OPC-UA compliant server, such as the one the OPC-UA module provides. To create a new connection, go to go `OPC Connections>Servers` and click "Create new OPC Server Connection". Select "OPC-UA Connection" from the list. OPC-UA connections communicate via TCP/IP so configuration is relatively straight-forward.

Main

Name	A name used to identify this connection.
Description	Short description of this connection, i.e. "Connection to plant floor."

Connection Settings

Host	The host name or IP address of server. If the OPC-UA module is running on the same computer you are configuring this connection on then "localhost" will likely be sufficient.
Port	The port the OPC-UA server is running. The OPC-UA module defaults to running on port 4096 but can be changed on the OPC-UA module settings page.
Security Policy	A Security Policy is a set of security algorithms that will be used together during a security handshake. The OPC-UA server this connection is intended for must support the chosen security policy.
Message Security Mode	The Message Security Mode and the Security Policy specify how to secure messages sent via this connection. None - No security is applied. Sign - Messages are signed but not encrypted. Sign And Encrypt - Messages are signed and encrypted.

© 2011 Inductive Automation

Enabled	A connection can be set to Enabled or Disabled. Disabled connections have their settings preserved but no actual connection is made and the server will not show up in the OPC Server Browser.

Authentication
If a username and password are specified then they will be used as a user identity token when connecting to the specified OPC-UA server.

The internal OPC-UA server provided by the OPC-UA module uses an Ignition security profile to govern who can connect to it. This can be configured in the OPC-UA module settings section.

Clustered OPC-UA Connection

A Clustered OPC-UA Connection should be used only when clustering is enabled. It has the same settings and aside from internal clustering logic functions the same as the regular OPC-UA Connection. The difference is that all cluster nodes will relay their OPC-UA connections through the master, meaning that connections all originate from a single point.

3.8.2.2 Connecting to OPC Classic (COM)

OPC-DA Connection

The OPC-DA connection, provided by the OPC-COM module, provides a link to a classic, COM-based OPC server. The connection supports OPC-DA versions 2 and 3. To create a new connection, go to go `OPC Connections>Servers` and click "Create new OPC Server Connection". Select "OPC-DA" from the server connection list. A list of OPC-DA servers installed on the machine will be presented. Select the desired server to create the connection.

Connections to OPC servers will be held open while the Ignition gateway is running. All subscriptions to the server will use the same connection.

Connecting to Remote Servers

While not recommended due to the inherent insecurity and instability of DCOM, the OPC-COM module offers the ability to connect to remote OPC servers over DCOM. To create a remote connection, start by following the instructions above, but select "Other Server" when selecting an OPC server to connect to.

On the following screen, fill out the basic information. The *ProgId* is not crucial, as the server's specific *CLSID* (unique ID) will be used to connect.

Expand the advanced options by selecting the "advanced options" check box. Fill in the options:

Remote Server
Select this box to specify that this is a server not located on the local machine, that will use a DCOM connection.

Host Machine
The computer name or IP address of the machine running the remote server.

CLSID
When connecting to a remote machine, it is currently required that you manually enter the *CLSID* of the server. This id can be found in the registry of the machine hosting the server under:

```
HKEY_CLASSES_ROOT\OPCServerName\CLSID
```

© 2011 Inductive Automation

3.8.3 OPC Quick Client

The OPC Quick Client can be accessed from under the "OPC Connections" section of the Ignition Gateway. It allows for quick, simple testing of any devices connected to the server.

You can browse by expanding tree nodes and read/write to tags by clicking on the [r] and [w] buttons next to those tags.

Subscriptions can be made by clicking on the [s] button. Clicking on the "enable live values" link will automatically refresh subscriptions and show live value changes (if there are any).

3.8.4 Ignition OPC-UA Server

3.8.4.1 OPC-UA Server Settings

Authentication

Authentication Profile	The Authentication Profile that the OPC-UA module will use to authenticate incoming connections against.
Allowed Roles	Roles within the given Authentication Profile that are allowed to connect to the server. Multiple roles should be separated by a comma, for example, Administrator,user,manager.
Allow Anonymous Access	If checked this will allow anonymous connections to the server.

Server

Server Port	The port the OPC-UA module runs on.
Endpoint Address	Overrides the address that will be returned in the endpoint URL during a GetEndpointsRequest from a client. This is useful if the server machine has a VPN connection or multiple adapters and is returning the wrong address.

3.8.4.2 Adding a New Device

To add a new Device go to the "Devices" section of the OPC-UA module configuration in the *Ignition* Gateway. Click "Add a Device..." and you will be taken to a page where you can select the driver to use. Choose your driver and click the "Next" button.

"General" settings common to all devices are as follows:

Device Name	The user-defined name for this Device. The name chosen will show up in OPC Item Paths and under the "Devices" folder on the OPC-UA server.

© 2011 Inductive Automation

	The Device Name must be alphanumeric.
Browse Timeout	Amount of time (in milliseconds) before a browse operation on this device times out.
Read Timeout	Amount of time (in milliseconds) before a read operation on this device times out.
Write Timeout	Amount of time (in milliseconds) before a write operation on this device times out.
Enable Device	Only devices that are enabled will appear in the "Devices" folder of the OPC-UA server and thus have their tags available for use.

3.8.4.3 Verifying Device Connectivity

Device connectivity can be verified either in the "Devices" section under the OPC-UA Server section, The Overview section of the Status page in the "Device Connections" bubble, or in the OPC-UA Server section of the Status page.

3.8.4.4 Drivers

3.8.4.4.1 Allen Bradley Drivers

3.8.4.4.1.1 ControlLogix 5500

ControlLogix Connectivity Settings

Hostname	The Hostname value is the IP Address of the ControlLogix Ethernet module (1756-ENET) to route through to connect a ControlLogix processor. EthernetIP protocol on TCP port 44818 (0xAF12) is used to communicate to ControlLogix processors.
Communication Timeout	After sending a request to the ControlLogix processor, the Communication Timeout setting is the amount of time in mSec to wait for a response before treating it as a failure.
Browse Cache Timeout	When the data table layout is read from the ControlLogix processor, the Browse Cache Timeout value is the amount of time in mSec to cache the results.
Slot Number	The Slot Number value is the zero based ControlLogix chassis slot number of the ControlLogix processor to connect to.

Supported ControlLogix Connection Methods

ControlLogix 5500 connected through 1756-ENET/A or 1756-ENET/B.

3.8.4.4.1.2 MicroLogix 1100/1400

MicroLogix 1100/1400 Connectivity Settings

Hostname	The Hostname value is the IP Address of the MicroLogix 1100 processor, MicroLogix 1400 processor or 1761-NET-ENI Ethernet interface. EthernetIP protocol on TCP port 44818 (0xAF12) is used to communicate to the listed devices.
Communication Timeout	After sending a request to the MicroLogix processor, the Communication Timeout setting is the amount of time in mSec to

© 2011 Inductive Automation

	wait for a response before treating it as a failure.
Browse Cache Timeout	When the data table layout is read from the MicroLogix processor, the Browse Cache Timeout value is the amount of time in mSec to cache the results.

Supported MicroLogix Connection Methods

MicroLogix 1100 and 1400 direct
MicroLogix 1100 and 1400 connected through 1761-NET-ENI
MicroLogix 1100/1400 connected through Spectrum Controls WebPort 500

Note: MicroLogix 1200 and 1500 are not fully supported. Browsing is not available on these devices.

3.8.4.4.1.3 PLC-5

PLC-5 Connectivity Setting

Hostname	The Hostname value is the IP Address of the PLC-5 processor. The protocol that the PLC-5 processor supports is automatically detected. It will use either CSP protocol on port 2222 (0x8AE) or EthernetIP protocol on port 44818 (0xAF12).
Communication Timeout	After sending a request to the PLC-5 processor, the Communication Timeout setting is the amount of time in milliseconds to wait for a response before treating it as a failure.
Browse Cache Timeout	When the data table layout is read from the PLC-5 processor, the Browse Cache Timeout value is the amount of time in milliseconds to cache the results.
Connection Path	The Connection Path value is used to define the route of the PLC-5 processor to connect to. Currently routing through the ControlLogix Ethernet Communication Interface Module (1756-ENET) to the ControlLogix Data Highway Plus-Remote I/O Communication Interface Module (1756-DHRIO) and on to a PLC-5 processor of the DH+ network is supported.

More Information On Connection Path

The Connection Path format contains 4 numbers separated by commas. The first number is always 1 and tells the 1756-ENET module to route through the backplane. The second number is the slot number of the 1756-DHRIO module of the DH+ network the PLC-5 processor is connected to. The third number is the channel of the 1756-DHRIO module that the PLC-5 processor is connected to. Use 2 for channel A and 3 for channel B. The final and fourth number is the DH+ node number. This number is in octal and is the same as configured in the PLC-5 processor. See the ControlLogix Ethernet Communication interface Module User Manual for more information.

Connection Path Format: 1,<1756-DHRIO slot number>,<1756-DHRIO channel>,<DH+ node number>

The valid range for the 1756-DHRIO slot number is between 0 and 16 but depends on the chassis size. The 1756-DHRIO channel is either 2 for channel A or 3 for channel B.

© 2011 Inductive Automation

The DH+ node number range is from 00 to 77 octal.

Supported PLC-5 Connection Methods

PLC-5 L/20E, L/40E, L/80E direct
All PLC-5 processors connected through DH+ via the 1756-DHRIO module.

3.8.4.4.1.4 SLC 505

SLC Connectivity Settings

Hostname	The Hostname value is the IP Address of the SLC processor. The protocol that the SLC processor supports is automatically detected. It will use either CSP protocol on port 2222 (0x8AE) or EthernetIP protocol on port 44818 (0xAF12).
Communication Timeout	After sending a request to the SLC processor, the Communication Timeout setting is the amount of time in milliseconds to wait for a response before treating it as a failure.
Browse Cache Timeout	When the data table layout is read from the SLC processor, the Browse Cache Timeout value is the amount of time in milliseconds to cache the results.
Connection Path	The Connection Path value is used to define the route of the SLC processor to connect to. Currently routing through the ControlLogix Ethernet Communication Interface Module (1756-ENET) to the ControlLogix Data Highway Plus-Remote I/O Communication Interface Module (1756-DHRIO) and on to a SLC processor of the DH+ network is supported.

More Information On Connection Path

The Connection Path format contains 4 numbers separated by commas. The first number is always 1 and tells the 1756-ENET module to route through the backplane. The second number is the slot number of the 1756-DHRIO module of the DH+ network the SLC processor is connected to. The third number is the channel of the 1756-DHRIO module that the SLC processor is connected to. Use 2 for channel A and 3 for channel B. The final and fourth number is the DH+ node number. This number is in octal and is the same as configured in the SLC processor. See the ControlLogix Ethernet Communication interface Module User Manual for more information.

Connection Path Format: 1,<1756-DHRIO slot number>,<1756-DHRIO channel>,<DH+ node number>

The valid range for the 1756-DHRIO slot number is between 0 and 16 but depends on the chassis size.
The 1756-DHRIO channel is either 2 for channel A or 3 for channel B.
The DH+ node number range is from 00 to 77 octal.

Supported SLC Connection Methods

SLC505 direct
SLC505, SLC504, SLC503 connected through 1761-NET-ENI
SLC504 connected through 1756-DHRIO
SLC505, SLC504, SLC503 connected through Spectrum Controls WebPort 500

© 2011 Inductive Automation

3.8.4.4.2 Simulator Drivers

3.8.4.4.2.1 Generic Simulator

The generic simulator provides a variety of tags that offer different data types and value generation styles. For example, there are ramps, sine waves, and random values. Additionally, there is a set of static writable tags whose values will persist while the device is running.

There are no configurable settings for the generic simulator.

Simulator tags

ReadOnly - static values that do not change for read only purpose

ReadOnlyBoolean1 - false

ReadOnlyBoolean2 - true

ReadOnlyShort1 - 1

ReadOnlyShort2 - 2

ReadOnlyInteger1 - 1

ReadOnlyInteger2 - 2

ReadOnlyLong1 - 1

ReadOnlyLong2 - 2

ReadOnlyFloat1 - 1.1

ReadOnlyFloat2 - 1.2

ReadOnlyDouble1 - 1.1

ReadOnlyDouble2 - 1.2

ReadOnlyString1 - "ABCDEFG"

ReadOnlyString2 - "ZYXWVUT"

Writeable - static values that you can read/write to, initial values below

WriteableBoolean1 - false

WriteableBoolean2 - false

WriteableShort1 - 0

WriteableShort2 - 0

WriteableInteger1 - 0

WriteableInteger2 - 0

WriteableLong1 - 0

WriteableLong2 - 0

WriteableFloat1 - 0

WriteableFloat2 - 0

WriteableDouble1 - 0

WriteableDouble2 - 0

WriteableString1 - "" (empty string)

WriteableString2 - "" (empty string)

© 2011 Inductive Automation

Random - Random values updating at some rate, they follow Java Random(rate) - rate is the seed
RandomBoolean1 - 10 sec
RandomShort1 - 5 sec
RandomInteger1 - 1 sec
RandomLong1 - 2 sec
RandomFloat1 - 10 sec
RandomDouble1 - 10 sec

Sine - Different sine waves with frequency, amplitude and offset (listed in that order)
Sine1 - 0.1, 100.0, 0.0
Sine2 - 0.01, 50.0, -25.0
Sine3 - 0.02, 10.0, 10.0
Sine4 - 0.04, 100.0, 0.0
Sine5 - 0.08, 100.0, 0.0

Ramp - Ramp signals starting from 0 going up to some value at the specified rate. When they reach their upper limit, they are reset to zero.
Ramp1 - 0 - 100 @ 10 ms
Ramp2 - 25 - 75 @ 100 ms
Ramp3 - 0 - 100 @ 50 ms
Ramp4 - 0 - 100 @ 25 ms
Ramp5 - 0 - 100 @ 12.5 ms

Realistic - Values determined by adding a random number (between -1 and 1) to the current value.
Realistic1 - -50 - 50 @ 500 ms
Realistic2 - -50 - 50 @ 1000 ms
Realistic3 - -50 - 50 @ 1500 ms
Realistic4 - -50 - 50 @ 2000 ms
Realistic5 - -50 - 50 @ 2500 ms

3.8.4.4.2.2 Allen Bradley SLC Simulator

The SLC simulator driver creates a simple device whose address structure mimics a basic SLC structure. There are currently no configurable parameters.

3.8.4.4.3 Modbus Drivers

3.8.4.4.3.1 Modbus Ethernet

The generic Modbus driver allows the Ignition OPC-UA server to communicate with any device that supports Modbus TCP protocol.

The Modbus driver can connect directly to devices that support Ethernet communications. It can also

© 2011 Inductive Automation

connect to Modbus devices through a gateway. It is important to only add one Modbus device in the Ignition Device List per IP address. When communicating to multiple Modbus devices through a gateway each with a unique unit ID, either include the unit ID in the Modbus specific address or set it in the address mapping for the device. See below for more information of each method.

Properties

Hostname	The Hostname value is the IP Address of the Modbus device.
Communication Timeout	After sending a request to the Modbus device, the Communication Timeout setting is the amount of time in milliseconds to wait for a response before treating it as a failure.
TCP Port	The TCP port to use when connecting to a Modbus device. The Modbus TCP port specified in the Modbus specification is 502, but it can be changed to a different port.
Maximum Holding Registers per Request	Some Modbus devices cannot handle the default of requesting 125 Holding Registers in one request. To accommodate this limitation change this setting to the maximum number of Holding Registers the device can handle.
Maximum Input Registers per Request	Some Modbus devices cannot handle the default of requesting 125 Input Registers in one request. To accommodate this limitation change this setting to the maximum number of Input Registers the device can handle.
Maximum Discrete Inputs per Request	Some Modbus devices cannot handle the default of requesting 2000 Discrete Inputs in one request. To accommodate this limitation change this setting to the maximum number of Discrete Inputs the device can handle.
Maximum Coils per Request	Some Modbus devices cannot handle the default of requesting 2000 Coils in one request. To accommodate this limitation change this setting to the maximum number of Coils the device can handle.
Use Zero Based Addressing	The Modbus specification states that Modbus addresses are to be zero based. Meaning Modbus addresses start at 0 instead of 1 and to read a value from Modbus address 1024, 1023 is sent to the device. When connecting to devices that do not adhere to zero based addressing, make sure this option is not selected. This will cause 1024 to be sent to the device to read Modbus address 1024.
Reverse Numeric Word Order	When reading and writing 32bit values from/to a Modbus device, the low word comes before the high word. By checking this option, the high word will come before the low word. The Modbus specification does not include a section for reading and writing 32bit values and as a result device manufacturers have implemented both methods.
Reverse String Byte Order	When reading and writing string values from/to a Modbus device, the low byte comes before the high byte. By checking this option the high byte will come before the low byte. If reading a string value from a device should read ABCD but BADC appears in Ignition then check this option.
Right Justify Strings	Strings stored in a Modbus device may contain leading spaces or trailing spaces. This can produce unwanted results so that Modbus driver removes spaces or zeros when reading string values. By default, left justify string handling will be used when reading and writing strings. By checking this option, right justify string handling will be used.

Modbus Specific Addressing

© 2011 Inductive Automation

Per the Modbus protocol specification there are four basic types of addresses that can be read from a device. These include Holding Registers (read/write 16 bit words), Input Registers (read only 16 bit words), Coils (read/write bits) and Discrete Inputs (read only bits associated with device input points).

Modbus Specific Addresses can be manually entered into the OPC Item Path of an OPC Tag using the following designators followed by the Modbus address.

`HR` for Holding Register

`IR` for Input Register

`C` for Coil

`DI` for Discrete Input

Because some devices that support Modbus protocol store data in BCD format, there are two additional designators. These designators will convert the data from BCD format to decimal when reading data from the device and convert data from decimal to BCD when writing to the device.

`HRBCD` for Holding Register with BCD conversion

`HRBCD32` for 2 consecutive Holding Registers with BCD conversion

`IRBCD` for Input Register with BCD conversion

`IRBCD32` for 2 consecutive Input Registers with BCD conversion

To accommodate other data encoding commonly used by Modbus supported devices, the following designators are available for Modbus specific addressing.

`HRF` for 2 consecutive Holding Register with Float conversion.

`HRI` for 2 consecutive Holding Register with 32 bit integer conversion.

`HRUI` for 2 consecutive Holding Register with 32 bit unsigned integer conversion.

`HRUS` for Holding Register with 16 bit unsigned integer conversion.

`IRF` for 2 consecutive Input Register with Float conversion.

`IRI` for 2 consecutive Input Register with 32 bit integer conversion.

`IRUI` for 2 consecutive Input Register with 32 bit unsigned integer conversion.

`IRUS` for Input Register with 16 bit unsigned integer conversion.

To read or write string values from/to a Modbus device, the following designation is available for Modbus specific addressing.

`HRS` read or write consecutive Holding Registers as a string value.

Note that there are 2 characters for each word and the order of which character comes first is controlled by the Reverse String Byte Order device setting as described above. Because two characters are stored in a word, the string length must be an even number of characters.

```
HRS FORMAT: HRS<Modbus address>:<length>
```

Examples:

```
[DL240]HR1024 Read 16bit integer value from Holding Register 1024.
[DL240]HRBCD1024 Read 16bit BCD value from Holding Register 1024.
[DL240]IR512 Read 16bit integer value from Input Register 512.
[DL240]C3072 Read bit value from Coil 3072.
[DL240]IR0 Read 16bit integer value from Input Register 0.
```

© 2011 Inductive Automation

```
[DL240]HRS1024:20 Read 20 character string value starting at Holding Register 1024.
```

The Modbus unit ID can also be specified by prepending it to the Modbus address. For example, to access Modbus unit ID 3 and read `HR1024` the full OPC path is `[DL240]3.HR1024`.

The Modbus specification does not support bit level addressing but it can be specified in the OPC Item Path. Please note that this only applies to reading bits of words and does not apply to writing bit values.

Example:

```
[DL240,bit=7]HR1024
```

Address Mapping

Because it can be very tedious manual entering OPC Tag information one-by-one, the driver has an address mapping feature. This allows entering blocks of common addresses and the driver will create the individual addresses and display them in the OPC browser.

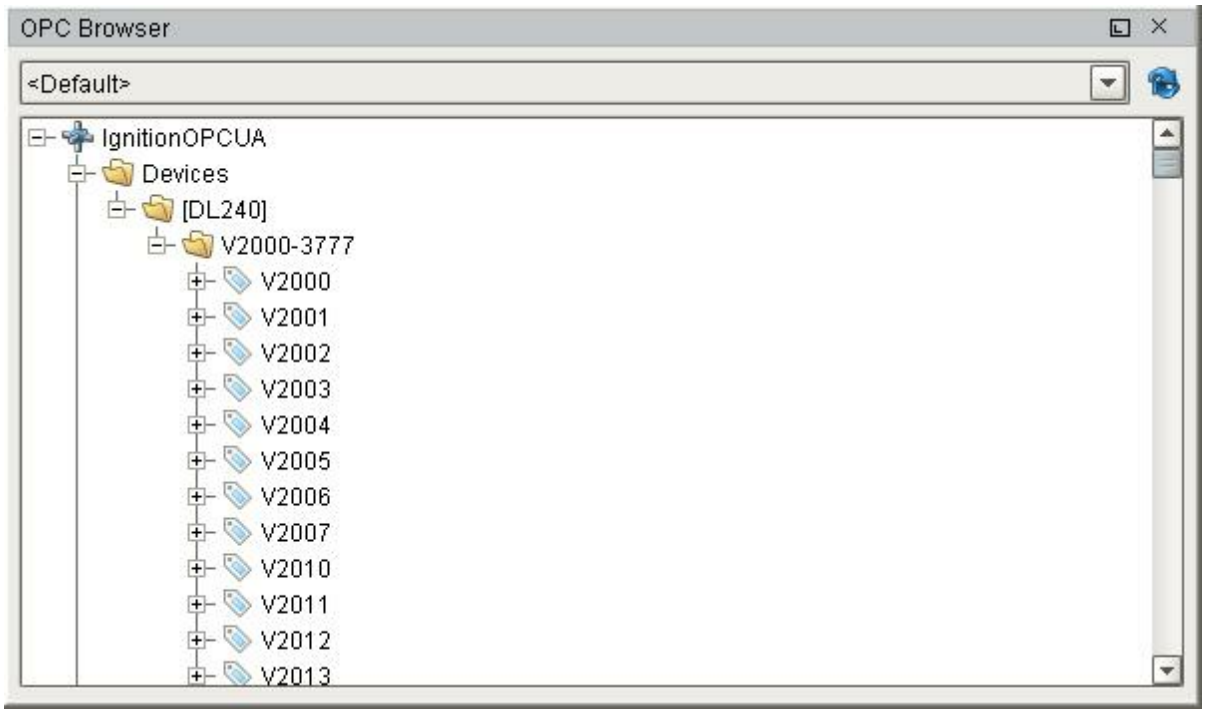
Another benefit of address mapping is the addresses inside a device can have a different numbering scheme than the Modbus address. The Direct Automation DL240 is a perfect example of this. Address V2000, capable of holding a 16 bit integer, is Modbus Holding Register 1024. In addition, the DL240 addressing is in octal meaning there are no 8 or 9s. The sequence of addresses go: V2000, V2001, V2002, V2003, V2004, V2005, V2006, V2007, V2010, V2011.... V3777. This is not very straight forward.

Below details how to map the DL240 address range V2000 to V3777 in octal to Modbus Holding Register addresses 1024 to 2047. Also, notice the Radix setting that in this example being equal to 8 causes the addresses to be in octal (also known as base 8).

Note that mappings for string data types cannot be entered. Strings can only be read or written using Modbus Specific Addressing. See above for more details.

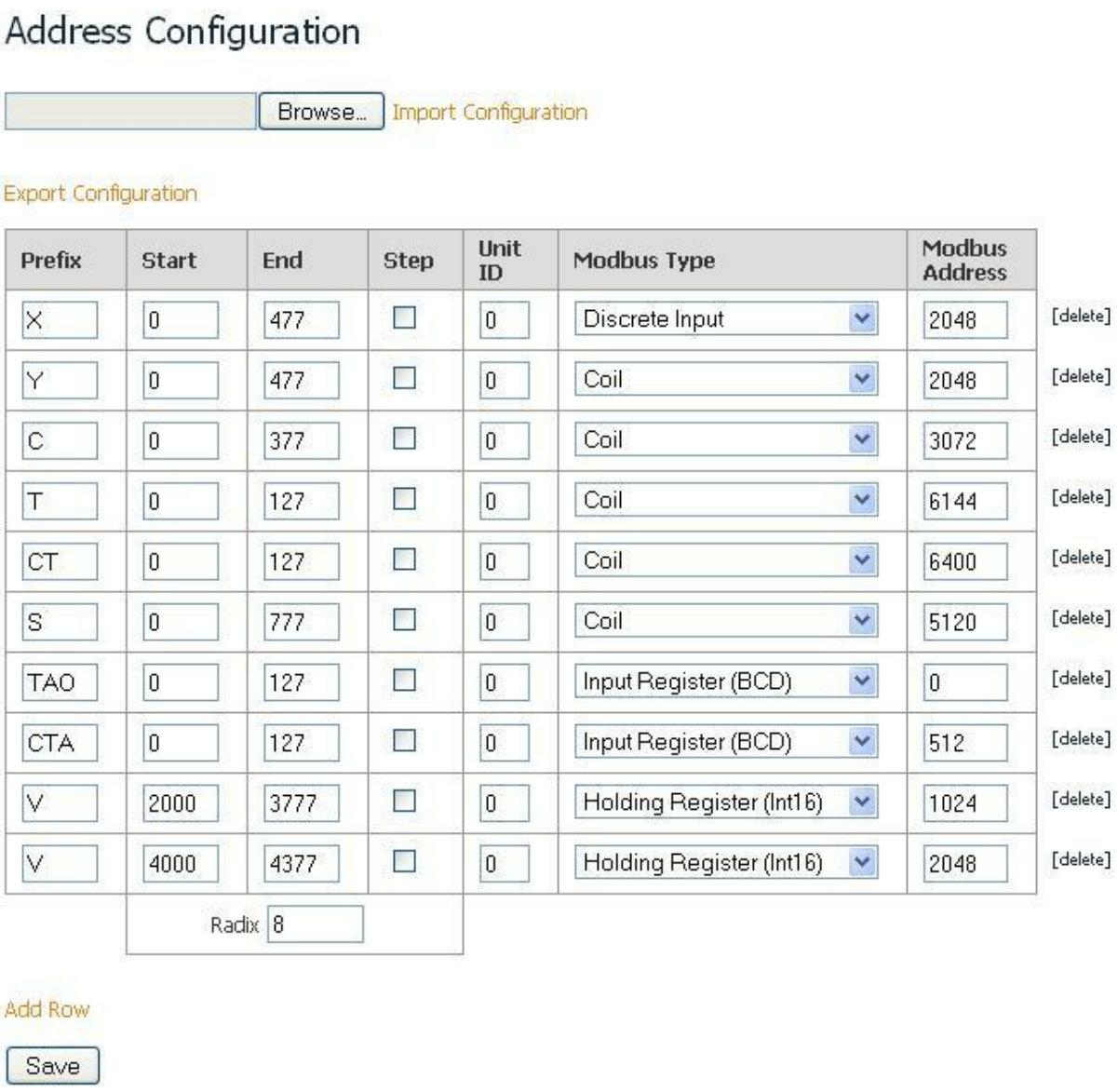
Once this mapping has been entered and saved, the OPC browser or the Quick Client will show all the DL240 addresses from V2000 to V3777 in octal.

© 2011 Inductive Automation



Example

© 2011 Inductive Automation


Address Configuration
Browse...
Import Configuration
Export Configuration
Prefix
Start
End
Step
Unit ID
Modbus Type
Modbus Address
X 0 477 0 Discrete Input 2048 [delete]
Y 0 477 0 Coil 2048 [delete]
C 0 377 0 Coil 3072 [delete]
T 0 127 0 Coil 6144 [delete]
CT 0 127 0 Coil 6400 [delete]
S 0 777 0 Coil 5120 [delete]
TAO 0 127 0 Input Register (BCD) 0 [delete]
CTA 0 127 0 Input Register (BCD) 512 [delete]
V 2000 3777 0 Holding Register (Int16) 1024 [delete]
V 4000 4377 0 Holding Register (Int16) 2048 [delete]
Radix 8
Add Row
Save

This shows mapping for all of the DL240 addressing.

When communicating to multiple devices through a Modbus gateway where the gateway only has one IP address, it is not recommended to add multiple Modbus devices with the same IP address. Only one Modbus device should be added to the Ignition OPC-UA Server device list for the gateway and to specify the different unit IDs in teh address mapping. The unit ID is specified for each entry in the address mapping for the Modbus device. Notice in the example address mapping below, that the Prefix, Start, End, Modbus Type and Modbus Address can be the same for two entries provided that the Unit IDs are different.

© 2011 Inductive Automation

Address Configuration

Browse... Import Configuration

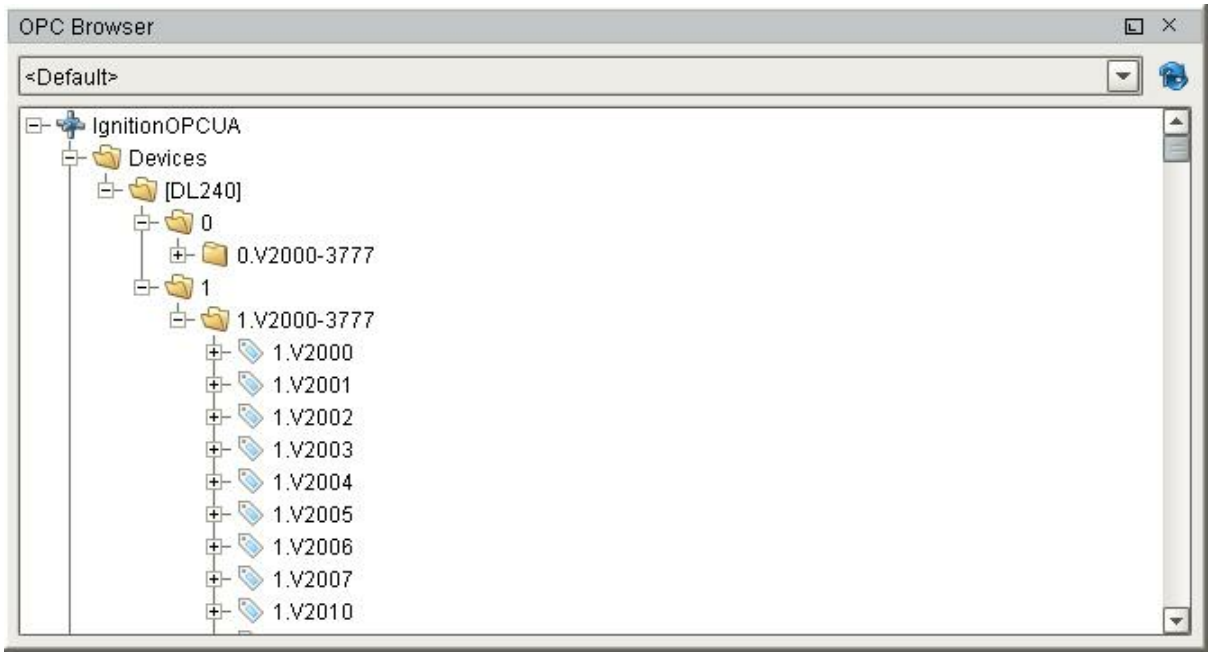
Export Configuration

Prefix	Start	End	Step	Unit ID	Modbus Type	Modbus Address	
V	2000	3777	☐	0	Holding Register (Int16)	1024	[delete]
V	2000	3777	☐	1	Holding Register (Int16)	1024	[delete]
	Radix 8						

Add Row

Save

Now when browsing the Modbus device, the unit ID will show as a folder and The OPC tag path will include the unit ID as shown below. This only happens when more than one unit ID is specified in the address mapping else the unit ID will be eliminated.



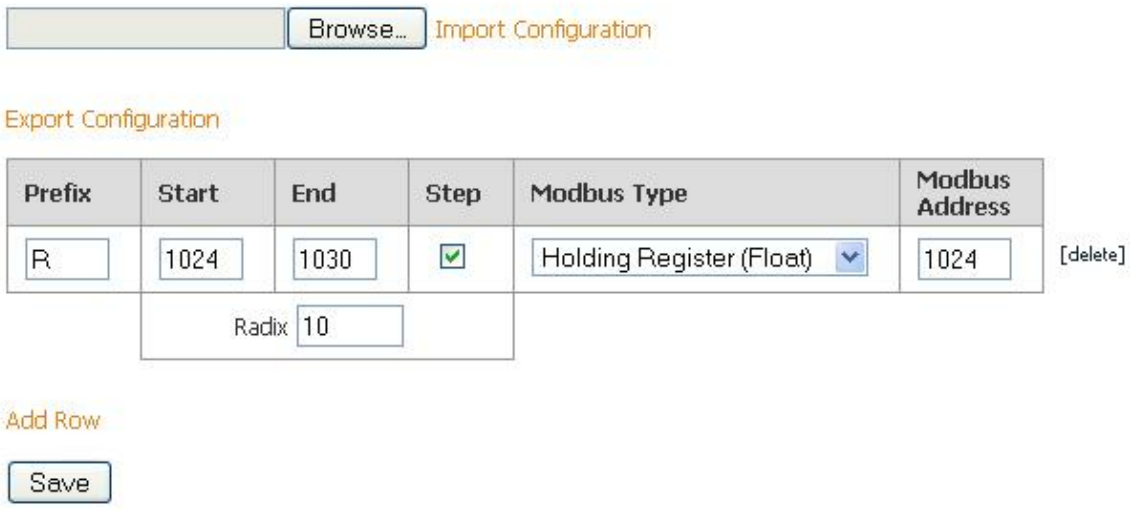
Modbus doesn't support reading and writing to any other memory types other than bits and 16 bit words. This is not very useful when reading from or writing to float point or 32 bit integers. To get around this the Modbus driver has been designed to read 2 consecutive 16 bit words and encode it into the desired data

© 2011 Inductive Automation

type.

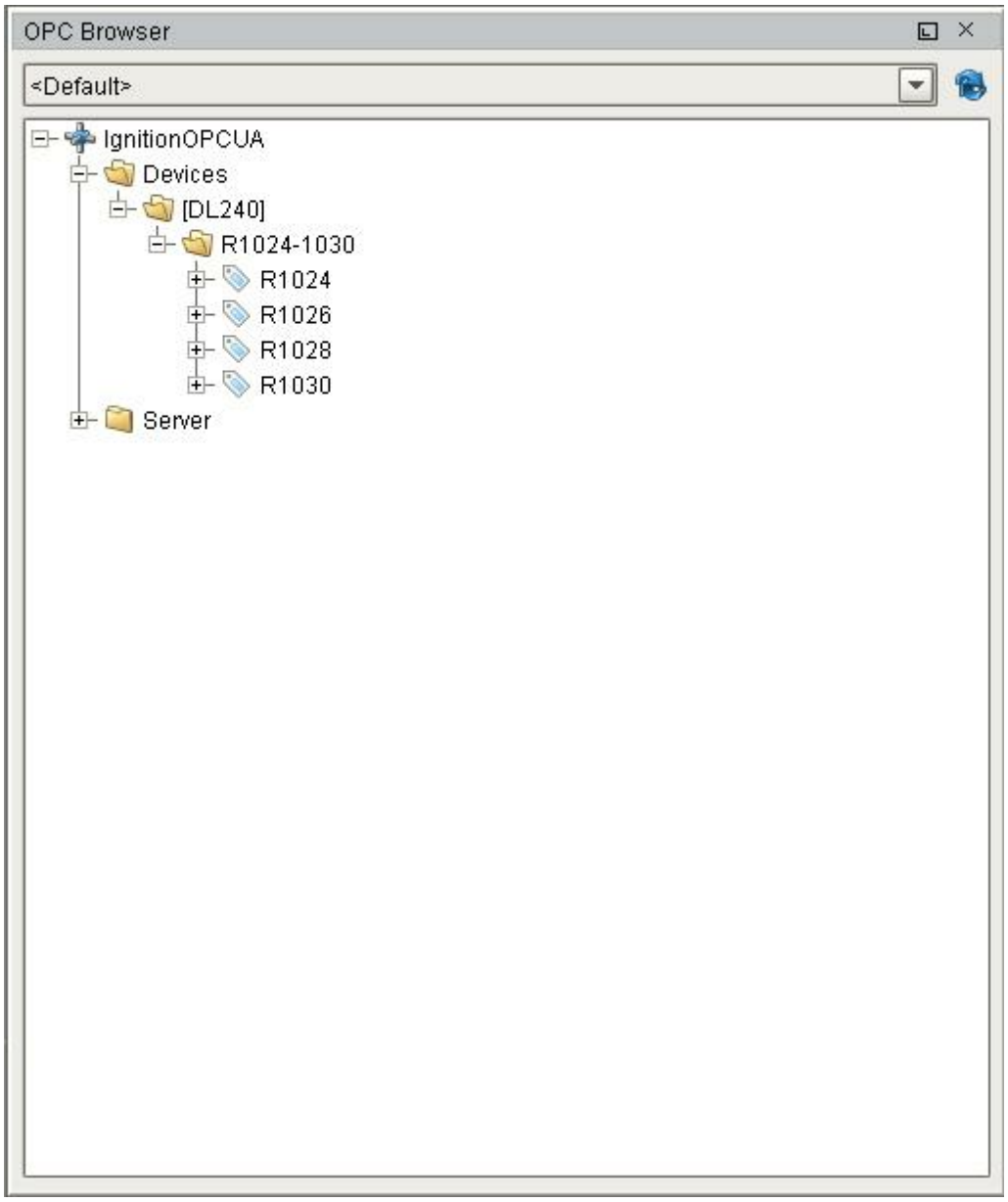
The Modbus address mapping below details how to map float point addresses starting at 1024 and ending at 1030. With the Step check box selected, the addresses on the Ignition side will be index by 2. In this case R1024, R1026, R1028 and R1030 will be created.

Because the Modbus Type of Holding Register (Float) is selected, the driver will read two consecutive 16 bit words and convert it to a floating point value. It will also index the Modbus Address by 2 for each entry. In this case, R1024 will read from Modbus addresses 1024 and 1025 and convert them into a floating point value. When writing, the reverse of converting a floating point value into two 16 bits words is done before sending them to the device.



This shows what appears in the OPC Browser. Notice that the numbering is index by two and that it matches the Modbus address. With some devices, this will allow the addresses appearing in the OPC Browser to match the addresses in the device.

© 2011 Inductive Automation



Import / Export Address Mapping

The mapping configuration can be exported to a comma separated values (csv) file. The csv file can later be imported in other Ignition installations or like devices.

3.8.4.4.4 UDP and TCP Drivers

3.8.4.4.4.1 UDP and TCP

The UDP and TCP drivers are strictly passive listeners. The UDP driver is configured to listen to one or more ports on a given IP address. The TCP driver is configured to connect to one or more ports on a given IP address.

© 2011 Inductive Automation

Rules are configured that dictate how the incoming data is interpreted.

Structure in the Address Space

A device using the UDP or TCP driver appears in the Devices folder of the OPC-UA server with the name it was configured to use. Browse the device will yield one folder per port configured to listen on.

Browsing the port folder will yield 1 variable node containing the entire message received as well as an addition variable node per field configured.

A device configured with a field count of 4 would have 5 nodes total - 1 for the message and 4 for the fields.

Properties

General

Device Name	The name to give to the device using this driver. This is will appear in the Devices folder when browsing the OPC-UA server.
Browse Timeout	Amount of time before a browse operation times out.
Read Timeout	Amount of time before a read operation times out.
Write Timeout	Amount of time before a write operation times out.
Enable Device	Whether or not this device is currently enabled. Disabled devices will not make a connection attempt.

Connectivity

Ports	On the UDP driver this will be the port(s) to listen on. On the TCP driver this will be the port(s) to connect to. Separate multiple ports with a comma.
IP Address	On the UDP driver this will be the IP address to listen to. On the TCP driver this will be the IP address to connect to.

Message

Message Delimiter Type	Sets the method used to determine how much or what data length constitues a full "message". Packet Based - Assumes that whatever arrives in one packet, regardless if length or content, is the message. Character Based - Content is appended to a message buffer until the given character arrives, at which point the contents of the buffer are considered the message.

© 2011 Inductive Automation

	Fixed Size - Content is appended to a message buffer until some fixed number of bytes is received, at which point the contents of the buffer are considered the message.
Message Delimiter	If the message delimiter type is "Character Based" then this shall be the character used to identify a message. If the type is "Fixed Size" than this shall be the size used to identify a message.
Field Count	The number of fields within a message must be fixed. This property dictates how many fields will be present in each message. When the number of fields received does not match the designated count all nodes will receive quality BAD_CONFIG_ERROR.
Field Delimiter	The character(s) that are to be used as field delimiters. For example, the message "a|b|c|d" with a field delimiter of "|" (no quotes) would be split into four fields: "a", "b", "c", and "d". The field count would have to be set at 4.

3.8.4.4.5 Siemens Drivers

3.8.4.4.5.1 Overview

The Siemens Drivers Module provides support for connecting to S7-300, S7-400, and S7-1200 PLCs via TCP/IP using the S7 protocol.

For more information on configuring tags see Addressing.

3.8.4.4.5.2 Addressing

The S7 protocol does not support browsing so all tags from the device must be configured as SQLTags in the Ignition designer. This can be done either manually, as needed, or by importing in bulk using the SQLTags CSV import functionality.

When creating a tag, the "OPC Item Path" field will be of the format: "[device_name]address", without the quotes, where device_name is the name given to the device during configuration and address is an S7 address, the format of which is described in the following text.

Tag addresses are made up of three different components: *Area*, *DataType*, and *Offset*.

Area	Syntax
DataBlocks	`DBn,`
Inputs	`I`
Outputs	`O`
Flags	`M`
Timers	`T`
Counters	`C`

DataType	Syntax
`Bit`	`X`
`Byte`	`B`
`Char`	`C`
`Word`	`W`

© 2011 Inductive Automation

DataType	Syntax
Int	I
DWord	D
DInt	DI
Real	REAL
String	STRING or STRING.len

To form an address you combine the desired *Area* and *DataType* with an *Offset* into that area.

Examples:

```
      IB0               Byte at Offset 0 in the Inputs area.
      IW0               Word at Offset 0 in the Inputs area.
      DB500,DI8         DInt at Offset 8 in DataBlock 500.
      ISTRING24.50      A String of length 50 starting at offset 24 in the
Inputs area.
      IX20.3            Bit 3 of the Byte at Offset 20 in the Inputs area.
      T0                Timer at offset 0 (No DataType is specified for
Timers).
      C0                Counter at offset 0 (No DataType is specified for
Counters).
```

It is important to note that offsets are *absolute*. IW0 and IW1 share a byte. To get 2 consecutive, non-overlapping words you would need to address IW0 and IW2.

Bits

Bits are addressed by using the Bit *DataType* (X) and appending .bit to the end, where bit is in the range [0-7]. When addressing a Bit at a given offset, that offset is always treated as a Byte.

Strings

Strings are assumed to be in the S7 string format and have a max length of 210.

Timers

Timers are scaled up to a DWord and converted from S5 time format so they can represent the time in milliseconds without requiring any multipliers. When you write to a timer it is automatically converted from milliseconds into S5 time format for you. A *DataType* is not specified when accessing *Timers*.

Counters

Counters in the PLC are stored in BCD. The driver automatically converts to/from BCD for you and exposes any counter tags as UInt16 values. A *DataType* is not specified when accessing *Counters*.

3.9 SQLTags

3.9.1 SQLTags Configuration Overview

While the goal of SQLTags is to create an easy yet powerful tag model, the variety of options and terminology can sometimes make configuration confusing. The goal of this chapter is to provide a clear overview of the SQLTags landscape, and provide a clear guide to the configuration of various

© 2011 Inductive Automation

architectures. It will be useful to have a working knowledge of what SQLTags are and how they executed, described in the section What is a SQLTag? in the Project Design chapter.

SQLTags Providers and Drivers

At the highest level of configuration is the *SQLTag Provider*. A provider is a tag database (a collection of tags) and a name. An Ignition gateway can have any number of tag providers, and therefore the name is used to distinguish which provider a tag comes from.

Every provider holds tags, but not every provider is a SQLTag *driver*, or tag executor. Some tag providers simply make tags available to use, and the tag execution is performed by a different *driving provider* elsewhere. For example, the Database Provider is a SQLTag provider that simply watches a tag database stored in an external database. It does not execute tags, it only monitors the values of the tags present. Somewhere else there is a tag driver such as a Driving Datasource Provider or a legacy FactorySQL that is executing the tags and storing the values to the database.

Realtime vs. Historian Providers

As discussed above, all SQLTags reside in a tag provider and provide realtime values. Additionally, there is the concept of tag *historian* providers, which can store and query historical data for tags. Each tag can optionally have a historian provider assigned to it to whom it will report value changes for historical storage.

Realtime providers - Internal vs. External

The SQLTags "tag database", or how SQLTags tag configuration is stored, can take two forms. In the *external* form, tags are stored in a SQL database, outside of Ignition. For *internal* tags, the configuration is stored in the Ignition internal configuration file, next to windows, groups, etc. The two different storage mechanisms have different pro and cons, and so it is a good idea to acquaint yourself with each of them.

External SQLTags Providers

SQLTags were originally invented as a way to reliably bridge realtime status and control information through the database. Therefore, the external storage of SQLTags represents the original methodology, and in fact is why SQLTags have their name.

There are two possible external SQLTags providers in Ignition:

- Database Provider
- Database Driving Provider (provided by the SQL Bridge module)

The driving provider, as mentioned above, will execute tags as well as make available tags driven by other external drivers looking at the same database, such as other Ignition gateways using the driving provider, or legacy FactorySQL installations.

The primary benefit of external providers is that the data is stored in a central database, and is therefore accessible to other consumers besides just the local installation. In this way, it is possible to pull together data from geographically dispersed sites into a central location, and then make the data from each site available to all of the others.

The negative side to external providers is that all tag values must be written to the database and then polled for change, which is less efficient than holding them in memory as the internal provider does.

© 2011 Inductive Automation

Internal SQLTags Provider
As mentioned above, the internal SQLTags provider stores the tag configurations in the Ignition gateway. The tags cannot be accessed outside of that particular gateway, but in return the efficiency is much greater, as values do not need to be written to the database and polled. It is possible to create multiple internal providers per gateway.

3.9.2 Configuring Realtime SQLTags

Realtime SQLTags providers are configured in the gateway under `SQLTags > Realtime`.
After installation, the Ignition gateway will start with an internal provider defined. You can edit its name and settings by selecting *edit* to the right of its entry in the table, or create new providers by selecting *Create new realtime SQLTags provider* below the table.

3.9.3 SQLTags Realtime Provider Types

3.9.3.1 Internal Provider

The internal provider stores tags internally in the gateway, and executes them in memory. Static tag values are stored persistently, but otherwise no values are stored.

Settings

The internal tag provider only has one additional setting:

Default Database
The database connection that will be used anytime a tag needs to access the database, such as executing a SQL Query based Expression tag.

3.9.3.2 Database Provider

The *database provider* stores SQLTags in an open format in the specified database. This provider type **does not** execute tags- it simply models tags and monitors values driven by a different tag provider elsewhere, such as an Ignition gateway using the *database driving provider* or FactorySQL.

Settings

Database
The database connection where the SQLTags configuration is stored.

Poll rate
The rate (in milliseconds) at which to poll the tag database for changes in tag value or configuration.

Poll overlap
The amount of time to overlap polls by. If set to 0, the config scan will look for changes only since the last execution. However, on databases that do not support millisecond resolution or are performing less-than-optimally, this could result in missed changes. This setting will expand the window in order to avoid missing these changes.

3.9.3.3 Database Driving Provider

The *database driving provider* extends the *database provider* adding the ability to execute tags. The values will be stored to the SQLTags tag database in the specified database.

© 2011 Inductive Automation

Availability

The database driving provider is a feature of the SQL Bridge module. It is only available when the module is installed.

Settings

The driving provider shares most of the settings of the database provider. However, it adds some key properties for driving and browsing.

Driver name
The unique name of this driver. Since the tags are stored in a central database, there may be multiple providers and drivers operating on them. This name will be used to identify this driver instance, and the tags that it executes. While the driving provider will read all of the tags stored in the database, it will only execute those tags that are assigned to it.

Enable browsing (of OPC servers)
Allows remote browsing of the OPC servers available to this driver over TCP/IP. This allows other gateways to remotely browse and add tags assigned to this driver into the central database.

Browse port
The port to listen on for remote connections. This port must not be in use by any other entity on the machine. Also, each driving provider that wishes to support browsing must have its own port.

Browse address
The IP address/network name that remote gateways will use when browsing. Therefore, care must be taken that the address is available from the gateways that will try to connect. Also, since it is used for access by remote systems, it should *not* be the loopback address (*localhost* or 127.0.0.1).

The browse address and port will be stored in the SQLTags database so that other gateways can easily look them up. After the settings are configured, you should immediately see the driver name in the OPC browse list for the external provider on other systems looking at the same database.

Note: When using remote browsing, make sure that the local firewall has an exception for the port that is used to listen. Otherwise, remote machines will not be allowed to connect.

3.9.4 How SQLTags Historian Works

SQLTags Historian gives you the ability to quickly and easily store historical data for your tags, and provides efficient querying of that data. Options for partitioning and deleting old data help ensure that the system stays properly maintained with minimal extra work.

This section describes various aspects of how SQLTags Historian stores and queries data.

Historian Providers

The settings for SQLTags Historian providers are set in the gateway under SQLTags > Historian. Historian providers are automatically created and removed according to the configured database connections. By default they will be created with a one month partition size, and will not delete old data.

SQLTag Configuration and Historical Value Generation

The first step to storing historical data is to configure tags to report values. This is done from the

© 2011 Inductive Automation

Historian Properties page in the SQLTags editor in the designer. The properties include a historical scan class, that will be used to check for new values. Once values surpass the specified deadband, they are reported to the history system, which then places them in the proper store and forward engine.

Data storage

As mentioned, the historical SQLTags values pass through the store and forward engine before ultimately being stored in the database connection associated with the historian provider.

The data is stored according to its datatype directly to a table in the SQL database, with its quality and a millisecond resolution timestamp. The data is only stored on-change (of value or quality), thereby avoiding storage of the same unchanged value many times. The storage of scan class execution statistics ensures the integrity of the data.

While advanced users may change the table according to their database to be more efficient (for example, using a compressed engine), Ignition does not compress or encrypt the data in any way.

Querying

While the data is stored openly in the database, the format does not lend itself well to direct querying. Instead, the Ignition platform offers a range of querying options that offer a tremendous amount of power and flexibility. In addition to simple on-change querying, the system can perform advanced functions such as querying many tags from multiple providers, calculating their quality, interpolating their values, and coordinating their timestamps to provide fixed resolution returns.

Querying can be performed on tables and charts through the Historical Binding, and through scripting.

3.9.5 Configuring SQLTags Historian

SQLTag Historian providers are configured at `SQLTags > Historian`. A historian provider is created automatically for each database connection, and will be removed if the connection is removed. Although enabled by default, the providers won't interact with the database unless data is logged to them.

General Settings

Enabled

Whether the provider will be turned on and accept tag history data. If disabled, any data that is logged to the provider will error out and be quarantined by the store and forward engine, if possible.

Data Partitioning

SQLTags Historian can partition the data based on time in order to improve query performance. Partitions will only be queried if the query time range includes their data, thereby avoiding partitions that aren't applicable and reducing database processing. On the other hand, the system must execute a query per partition. It is therefore best to avoid both very large partitions, and partitions that are too small and fragment the data too much. When choosing a partition size, it is also useful to examine the most common time span of queries.

Partition Length and Units

The size of each partition. The default is one month. Many systems whose primary goal is to show only recent data might use smaller values, such as a week, or even a day.

© 2011 Inductive Automation

Data Pruning

The data prune feature will delete partitions with data older than a specific age. It is not enabled by default.

Enable
Monitor the partitions and drop those whose data is older than the specified age.

Prune Age and Units
The maximum age of data. As mentioned, the data is deleted by the partition, and could therefore surpass this threshold by quite a bit before all of the data in the partition is old enough to be dropped.

3.10 Security

3.10.1 Security Overview

Ignition uses the concept of *role-based security* throughout. Role-based security is the concept that each user may be assigned to various *roles*. Security policies are then defined in terms of these roles, rather than defined for specific users. This allows users to be reassigned, removed, and added without affecting the logic of the security policy.

The users and their roles are defined in *authentication profiles.* An Ignition Gateway may have many different authentication profiles defined, each governing the security of different aspects of the Gateway. For example, logging into the Gateway configuration web interface might be governed by one authentication profile, while the security for a project is governed by another.

There are many different types of authentication profiles that offer various features. For example, the *Internal* authentication profile offers the ultimate in ease-of-use: you simple define the users, their passwords, and the roles within the Ignition Gateway configuration web interface. In contrast, the *Active-Directory* authentication profile offers the power of integrating Ignition with a corporate security infrastructure. Users, passwords, and roles would be managed centrally by the IT department.

Security policies can be defined for many different parts of the system. For example:
You can alter the roles required to log into the Gateway *configuration section*
You can define roles required to write to or even read from a *SQLTag*
You can define roles required to view a *Component.*
You can access the security system in a *script* to restrict the operation of the script to authorized users.

3.10.2 Authentication Profile Types

3.10.2.1 Internal Authentication Profile

The internal authentication profile is very easy to use. This is the kind of authentication profile that you get when you first installed Ignition and had the default user/password combination of `admin` / `password`.

The internal authentication profile is very easy to use, and many projects continue to use it. You can define multiple internal authentication profiles. You can administer the users, passwords, and roles of an internal authentication profile within the Gateway's configuration interface through the special *manage users* link next to the profile in the `Configure > Security > Authentication` section of the Gateway.

© 2011 Inductive Automation

3.10.2.2 Database Authentication Profile

The database authentication profile uses an external database connection to find its users, their passwords, and their roles. When you first create a database authentication profile you can have it automatically create the appropriate tables through your database connection. This is usually a good idea, as it makes the setup very easy.

To administer the users and their roles, you'll have to interface directly with the database. This type of authentication profile is best when the ability to administer users from *within* a running client is a requirement.

3.10.2.3 Active Directory Authentication Profile

The active directory profile type will communicate with a Microsoft Active Directory server through the LDAP protocol. Administration of the users and roles must be done through Active Directory's management tools. This authentication profile is a good choice when integration with a corporate authentication scheme is a requirement.

To set up an active directory authentication profile, you must specify the host that is acting as your primary domain controller. You can also use a secondary domain controller in case the primary is unavailable. You'll also need to specify the name of the domain and credentials for the Gateway itself to use for authentication for when it queries the list of roles.

3.10.2.4 AD/Internal Authentication Profile

The active directory/internal hybrid profile type combines the internal profile for role management, but uses Active Directory for authentication. This means that for any username/password combination, Active Directory gets to decide whether that user is a valid user, and *if they are* considered valid, then the Ignition Gateway looks internally for their list of roles.

This type of authentication profile is very handy for projects that are required to integrate with IT's centrally managed security, but negotiating the management of *roles* with IT would be too cumbersome.

3.10.2.5 AD/Database Authentication Profile

The active directory/database hybrid profile type uses the database authentication profile for role management, but uses Active Directory for authentication. This means that for any username/password combination, Active Directory gets to decide whether that user is a valid user, and *if they are* considered valid, then the roles for that user are retrieved from an external database connection.

This type of authentication profile is very handy for projects that are required to integrate with IT's centrally managed security, but negotiating the management of *roles* with IT would be too cumbersome. The main reason one would choose this profile type over the AD/Internal profile is that by storing the roles in an external database, they can be managed outside of the Ignition Gateway's web configuration interface. Specifically, one can create screens using the Vision Module for role management, thus allowing security management from within a running Client.

3.10.3 Managing Users, Passwords, and Roles

How users, passwords, and roles are managed depends entirely on the type of authentication profile in question. There may be multiple authentication profiles defined. To know what kind of authentication profile is governing what, follow these simple steps:

1. To manage users and passwords for **logging into the Gateway Configuration section**, you'll need

© 2011 Inductive Automation

to see what authentication profile is currently set as the Gateway's authentication profile. You can check this under `Configuration > Gateway Settings` by looking at the *System Authentication Profile* field and the *Gateway Config Role(s)* field.
2. To manage users and passwords for **logging into the Designer**, you follow the same steps as in #1, except that you need to look at the *Designer Role(s)* field to see what roles are allowed to log into the designer.
3. To manage users and passwords for **logging into a Vision Client**, you go to the `Configuration > Projects` section. Look at the project in question and you can find its authentication profile listed there.
4. Now that you know what authentication profile you need to manage, you can find out what kind it is under the `Security > Authentication` section.

Now that you know what kind of authentication profile you're dealing with, you can learn how to manage the users, passwords, and roles for each.

1. **Internal** authentication profiles are the easiest to manage, because you do it all from within the Gateway's web configuration interface. Simply click on the *manage users* link to the right of the profile, and you can use the interface to add users, roles, and assign users to the various roles.
2. **Database** authentication profiles are typically used because you want to be able to manage the users and roles externally by reading and writing to an external database. Because this is the kind of thing a Vision Client does so well, this authentication profile type is often used for projects that require user management from within the Client application itself.
3. **Active Directory** authentication profiles are chosen because it is I.T.'s role to manage the users and groups. They have tools to do so, and this cannot be done from within Ignition.
4. **AD/Internal Hybrid** authentication profiles are a compromise between Active Directory and Internal profile types. *Users* and *passwords* are handled by Active Directory - a user must be able to authenticate correctly with the Active Directory service in order to log in. *Roles*, however, are managed internally, just like in the Internal profile type by clicking on the *manage users* link. To assign roles to a user, you add a user with the same username that Active Directory would authenticate with, and then assign any roles to them.
5. **AD/Database Hybrid** authentication profiles are a compromise between Active Directory and Database profile types. Just like the AD/Internal hybrid - active directory is used to handle the username and password verification. If a user authenticates correctly against active directory, their roles are retrieved from an external database connection, just like in the Database authentication profile type.

3.10.4 Enabling SSL Encryption

To enhance security in Ignition, you may opt to enable SSL encryption. This will affect all communication to and from the Gateway that is done over the HTTP protocol. This includes not only browsers interacting with the Gateway's web interface, but all Vision Client communication as well.

Turning on SSL will encrypt all data sent over HTTP. This protects your installation from anyone "snooping" the data as it passes over the network. This may be important if data transferred between the Gateway and clients is sensitive in nature. This also helps to thwart a security vulnerability known as "session hijacking".

To turn on SSL, navigate to the Configuration section of the Ignition Gateway's web interface. Use the left navigation menu to find the `Configuration > Gateway Settings` page. Here, check the "Use SSL" checkbox, and then press the "Save Changes" button.

After SSL is enabled, all clients and web browsers will be redirected to the SSL port if they try to use the

© 2011 Inductive Automation

standard HTTP port. By default, the SSL port is 8043. You may wish to change this to the standard SSL port of 443. To do this, follow the directions in Setting the Port.

3.11 Alerting

3.11.1 Alerting Overview

Alerting (also occasionally referred to as 'alarming'), is a core feature of the Ignition platform. Alerts are conditions that are evaluated with respect to a specific numeric datapoint. Most commonly, alerts are configured on a SQLTag or a Transaction Group item.

Any given datapoint can have multiple conditions that might cause it to be considered "in alert". For example, you might configure an analog tag to be in alert if its value exceeds 50.0, or you might configure a discrete tag to be in alert if its value non-zero. Analog values can have multiple alert states configured for them. Each alert state defines a numeric range where it is considered active, and has a name and a severity.

An alert state becomes *active* when the value of the datapoint falls within the range of the state. The alert state is said to *clear* when the datapoint moves outside of the range by at least the alert deadband, if the deadband is configured. When an alert state becomes active or clear, a message is generated and will be consumed by any configured alert storage profiles and alert notification profiles.

The job of an *alert storage profile* is to store the record of when an alert state for a datapoint became active, when it cleared, and whether or not it has been acknowledged. Typically, this is done by recording the event as a row in an external database.

An *alert notification profile* takes the messages from the alerting system and uses them to notify people of the event. This is typically done via sending an email. There are several types of alert notification profiles that provide different mechanisms for controlling how notifications are sent to various sets of users.

Information about configuring alerts conditions can be found in Alert Properties under SQLTag configuration.

Filters

Both notification and storage profiles offer the ability to filter alert messages on a few basic parameters. Multiple profiles of each type can be created and configured differently in order to filter out different sets of alerts, if desired. The three text based filters, *System, Path* and *State Name,* can include wildcard parameters * (any characters) and ? (any single character).

3.11.2 Alert Notification

Alert notification profiles hook into the alerting system in Ignition, listening for alert messages and sending them to people. This page describes the differences between the different types of alert notification profiles.

Basic Email Notification

The basic email notification profile simply sends all alert messages to a list of email recipients. The list of email recipients is configured in the Ignition Gateway's web configuration interface.

© 2011 Inductive Automation

Distribution List Email Notification

The distribution list email notification profile will email various groups of email recipients based upon logic that is evaluated for each message. For example, you could use it to send all alerts involving a compressor system to a certain group of maintenance personnel, while alerts involving product temperature might go to a group of QC personnel. Or, you could differentiate recipients based upon the time of day to notify the personnel in the correct shift.

The distribution list profile maintains a list of *contacts* and a list of *groups*. Each contact is a name, an email address, and a mapping of which groups the user belongs to. Each group defines an *expression,* which shares the syntax of other Ignition expressions, but can refer to properties of the alert. It is evaluated for each alert event that occurs, and messages that evaluate to TRUE are sent to the corresponding users.

3.11.3 Alert Storage

The alert storage profile allows you to create a historical log of alert events in the database. To create a new storage profile, select "Create new Alert Storage Profile" under `Alerting>Storage` in the gateway.

Database Storage Profile Settings

The key setting for the database storage profile is the *connection name*, which dictates the connection that will be used to store events.

Retention Period (days)
The number of days to store events. Older events will be deleted.

Database Connection
The connection to use for storing events.

Auto Create Table
The system will create the table in the database if necessary.

Table Name
The name of the table that will be used for alert event storage.

3.12 Redundancy

3.12.1 What is Redundancy?

Redundancy is an advanced feature of Ignition that provides a higher degree of fault-tolerance and protection from downtime due to machine failure. Using redundancy, two Ignition installations can be linked together, so that when one fails, the other takes over and continues executing. All of the clients connected will be redirected to the backup machine, and historical data will continue to be logged.

There are a variety of design decisions that come into play when setting up redundant systems, so it is important to understand the available options, and how the pieces of the system function in a redundant setting. This chapter will start with key terminology that will be used heavily, and will then proceed to explain how the main parts of the system function. It will then explain the various settings available, and will finish up with an examination of a few common setups.

Clustering vs. Redundancy, and previous versions of Ignition

Previous versions of Ignition contained a feature called *clustering* that was similar to redundancy in that it linked multiple systems, but different in terms of the goals it aimed to achieve. The primary goal of

© 2011 Inductive Automation

clustering was to provide a seamless platform for balancing many client connections across multiple servers. In the reality of the field, it was observed that client load was rarely a cause for concern. Ease of configuration and greater flexibility in creating redundant fail-over systems were larger concerns, and resulted in the switch to "redundancy".

Terminology

Here are some of the most common terms used in relation to clustering.

Activity Level
The activity level describes what the Ignition installation is currently "doing". A node in a redundant pair will operate at one of three levels: Cold, Warm, or Active. In "cold", the system is doing a minimal amount of work. In "Warm", the system is nearly running at full level, in order to switch over quickly. Both of these levels imply that the other node is currently active. In "active", the system is the primary system, responsible for running all sub-systems.

Node
A node is an Ignition installation, set to be part of the redundant pair. There can be a master node, and a backup node.

Active Node
The active node is the Ignition installation that is currently at the "active" level, and is responsible for running. It is also described occasionally as the "responsible node". It can be either the master or backup node, even when both are available. For example, if the backup node becomes active after the master node fails, and the master comes back up but is set to *manual recovery mode*, the backup will continue to be active until it fails or the user switches responsibility back to the master.

Master Node
The node that is responsible for managing the configuration state. It is also generally expected to be the active node when available, though this is dependent on settings. It is therefore import to separate the ideas of the *master node* and the *active node.*

Backup Node
The node that communicates with the master and takes over when that node is no longer available.

3.12.2 How Redundancy Works

Ignition redundancy supports 2-node systems. One node is considered the **master node**, and the other is the **backup node**. Both nodes share the same "state", or configuration. In other words, all projects, gateway settings, etc. are shared between the nodes. The management of the configuration is performed by the master node, and then replicated to the backup node.

Node Communication

The master and backup nodes communicate over TCP/IP. Therefore, they must be able to see each other over the network, through any firewalls that might be in place. All communication goes from the backup to the master node, by default on **port 8750**. Therefore, that port must allow TCP listening on the master machine. The port can be changed in the gateway redundancy settings page.

Configuration Synchronization

The master node maintains the official version of the system configuration. All changes to the system must be made on the master- the gateway on the backup will not allow you to edit properties. Similarly, the designer will only connect to the master node.

When changes are made on the master, they are queued up to be sent to the backup node. When the backup connects, it retrieves these updates, or downloads a full system backup if it is too far out of date.

© 2011 Inductive Automation

If the master node has modules that aren't present on the backup, they will be sent across. Both types of backup transfers- "data only" and "full"- will trigger the gateway to perform a soft reboot.

Runtime State Synchronization

Information that is only relevant to the running state, such as current alert states, is shared between nodes on a differential basis, so that the backup will take over with the same state that the master had. On first connection, or if the backup node falls too far out of sync, a full state transfer is performed. This information is light-weight and will not trigger a gateway restart.

Status Monitoring

Once connected, the nodes will begin monitoring each other for liveliness and configuration changes. While the master is up, the backup runs according to the **standby activity level** in the settings. When the master cannot be contacted by the backup for the specified amount of time, it is determined to be down, and the backup assumes responsibility. When the master becomes available again, responsibility will be dictated by the **recovery mode**, and the master will either take over immediately, or wait for user interaction.

System Activity

When a node is **active**, it runs fully, connecting to any configured OPC servers, and communicating with devices. When it is not active, its activity level is dictated by the settings, either **warm** or **cold**. In "warm" standby, the system will run as if it were active, with the exception of logging data or writing to devices, allowing for faster fail-over. In "cold" standby, the system does not subscribe to tag values, and does not communicate with any device. This allows the system to standby without putting additional load on the devices and network. Fail-over will take slightly longer, as tags must be subscribed and initialized.

Historical Logging

Historical data presents a unique challenge when working with redundancy, due to the fact that it is never possible for the backup node to know whether the master is truly down, or simply unreachable. If the master was running but unreachable due to a network failure, the backup node would become active, and would begin to log history at the same time as the master, who is still active. In some cases this is OK because the immediate availability of the data is more important than the fact that duplicate entries are logged, but in other cases, it's desirable to avoid duplicates, even at the cost of not having the data available until information about the master state is available. Ignition redundancy provides for both of these cases, with the **backup history level**, which can be either "Partial" or "Full". In "full" mode, the backup node logs data directly to the database. In "partial" mode, however, all historical data is cached until a connection is reestablished with the master. At that time, the backup and master communicate about the uptime of the master, and only the data that was collected while the master was truly down is forwarded to the database.

Client Fail-over

All Vision clients connect to the active node. When this system fails and is no longer available, they will automatically retarget to the other node. The reconnection and session establishment procedures are handled automatically, but the user will be notified that they have been transferred to a different node, so that they can notify the system administrator that they system may need attention.

3.12.3 Setting up Redundancy

To set up redundancy, you first must understand that both of the nodes share the exact same configuration state. This means that when a backup node joins to a master node, it will essentially

© 2011 Inductive Automation

download a backup of the master's state and restore that backup on itself. This fact leads to a couple of observations:

1. It is best to start with a fresh install for the backup node. The current configuration of the backup node will be overwritten, so make sure that it does not contain anything valuable in it when enabling redundancy.
2. All system configuration relative to the master node must also resolve on the backup node. For example, OPC-UA connections and database connections must use addresses that resolve from both nodes, any OPC-COM servers must be installed and configured identically on both nodes, etc.

With that in mind, setting up redundancy is fairly simple. Follow these steps to set up your redundant pair:

1. **Turn off firewalls between the redundancy nodes.** Redundant systems need TCP connectivity between each other on a variety of ports. Turning off software firewalls or adding special exception rules for each others' addresses is required. Specifically, The master node must be able to receive data on TCP/IP port **8750** (changeable in settings), and the backup node must be able to send outgoing data on that port.
2. **Configure the master node.**
   2.1. Set mode to 'Master' under the **Configuration > Redundancy** in the gateway configuration.
   2.2. It is advisable to turn off 'Auto-detect network interface' and to manually specify the address of the NIC (network interface card) to use for communication.
   2.3. The addition settings are described in the next section, redundancy settings.
3. **Configure the backup node**
   3.1. On the desired backup system, set the mode to 'Backup'.
   3.2. Under 'Backup Node Settings', specify the address of the master node. Also verify that the port is correct under 'Network Settings'.
   3.3. After saving, the system will connect to the master and will download a system backup, which will trigger a restart. After the restart is complete, the backup node should now be synchronized and in communication with the master.
4. **Verify Redundancy Setup with the System Map.** When you go to the status section of the gateway, the system map should show both connected nodes and should show their current states.

3.12.4 Redundancy Settings

All redundancy settings are configured in the Ignition configuration gateway under **Configuration>Redundancy**. Most settings are used by both the master and backup nodes, with their individual settings broken out into separate categories.

It is important to know that while the full system configuration is shared between nodes, **redundancy settings are not shared between nodes.** Therefore, it is perfectly acceptable to have different values for the same settings on the two nodes. For example, it is possible to have a different Standby Activity Level on both nodes, and, of course, the network settings will often be different.

Redundancy Settings

Mode

Independent - Redundancy is not enabled and this Ignition system runs as an independent node.

Master - This is the master node, who listens for a connection from the backup node, and is in charge of managing system synchronization.

© 2011 Inductive Automation

Backup - This is the backup node, who will connect to the master and receive system updates.

Standby Activity Level
How the node operates when it is not the "active" node.

Cold - perform minimal activities, do not connect to devices, etc. Purpose: minimize the load on the network and on devices.

Warm - Connect to devices, subscribe to tags and set up all executing objects. Purpose: minimize fail-over time.

Fail-over Timeout
The time, in milliseconds, before the opposite node is determined to be unavailable and this node takes over.

Startup Connection Allowance
The time, in milliseconds, to wait on initial startup for a connection to be established before making a decision on the node's activity level. This is used to prevent unnecessary switch over caused by a node starting as active, only to connect and find that the other node is active, resulting in one of the nodes being de-activated. It is important to note that this setting can interfere with the Master Recovery Mode- if the master is active, it will always request the backup to de-activate. If this setting is low, or 0, the master will always become active before connecting to the backup, and thus "manual recovery" will not be possible.

Network Settings

Port
For the master, the port to listen on. For the backup, the port to connect to on the master.

Auto-detect Network Interface
If `true`, the system will automatically select which network interface to use on the machine. If `false`, the system will bind itself to the interface of the specified address.

Network Bind Interface
The address to bind to if Auto-detect is `false`.

Auto-detect HTTP Address
When clients are launched, they are provided with a list of addresses that they may connect to. If this option is `true`, the list will be generated automatically. If `false`, they will be provided with the list specified.

HTTP Addresses
The list of addresses to give to the clients if auto-detect is turned off. These are the addresses that the clients will attempt to connect to, so the HTTP and HTTPS ports must match the configuration of the system in the Gateway Control Utility.

Master Node Settings

Recovery Mode
How the master acts when it connects to a backup node that is currently active.

Automatic - The master automatically takes back responsibility, and becomes active. The backup node goes to standby.

Manual - The backup node is allowed to stay active. The master will become active if the backup node fails, or if the user requests a switchover from the gateway configuration page.

Runtime Update Buffer Size
How many "runtime status updates" can be queued up in memory before the system stops tracking them and forces a full refresh. These updates represent information that the other node should have in order to have the same running state as the master when it's forced to take over. This is most often

© 2011 Inductive Automation

the values of static tags and the current alert state. Given that the update buffer is only used once the nodes are connected, the default value is usually fine, and only needs to be increased on systems that may have many alerts that change together, or many static tag writes.

Backup Node Settings

Master Node Address
The address where the master is located.

History Mode
How the backup node treats history when it is active.

Full - The system operates normally, and data is stored to the database.

Partial - The system caches all historical data until it can verify the time period that the master was actually down. This prevents the storage of duplicate data in the case that only the communication between the master and backup went down, resulting in a situation where both nodes ran as active for a period of time.

3.12.5 Database Considerations

Given that many parts of the Ignition system interact with the database, it's important to give some thought as to how it will be used when redundancy is turned on, and the different database architectures that are possible.

Ignition Database Requirements

When evaluating database architectures for use with Ignition, it's important to look carefully at how the system will use the database. Which pieces are critical? Which pieces are "optional", in that the system will continue to function while the database is down? Which pieces can operate in "read-only" mode if necessary?

Ignition uses the database for many purposes. Here are some common areas where they are used, and how availability can impact the system:

SQLTags
If using the default internal provider, SQLTags only rely on the database for tags which execute queries. These tags will error out if the database is unavailable, but the status and control functionality of the system will function on the whole. If using an *external provider,* which is housed in the database, the tags will no longer function. Therefore it's much more important to have a read-capable database connection available when using external tags.

History - SQLTags and Other
All history in Ignition goes through the store-and-forward system, meaning that it will be cached until the database is available. However, while the data is cached, it will be unavailable to view or analyze on the clients. Therefore, when looking at how the database should be set up, it is necessary to determine how crucial rapid-availability of the data is.

Alerting
The alert status system does not reside in the database, so it will continue to function if the connection is down. Alert log information will go through the store and forward system as history data.

Project screens
Almost all projects use database access for providing information on screens. These queries will error out as long as the database is unavailable. Screens that only use SQLTags (in an internal provider) will continue to function, so it would be beneficial to make a distinction between status screens and history screens, if a failover database is not used.

© 2011 Inductive Automation

Database Architectures

Single Shared Server

A single database server is used. The Ignition gateways will both use it, so it is expected to be available even when one of the nodes is not. For that reason, it almost always resides externally, on a separate server machine.

This arrangement is the easiest to use with Ignition. A single database connection configured on the master will be replicated to the backup, and both nodes will use the connection as necessary.

Clustered/Replicated Database Servers

There is a wide variety of capabilities supported by the different brands of database servers. To obtain fault-tolerance on the database front, it is usually necessary to have some sort of cluster/replication system in place. However, it can be very import to examine how Ignition is using the databases, and what capabilities the clustering solution provides.

For example, in many replication scenarios, the master database copies data to the backup. The backup can be used for read purposes, but new data inserted will not be replicated back to the master. Therefore, it is possible to have a failover connection to the backup database, so that clients will continue to receive data, but it would be necessary to run in *partial history mode*, so that the historical data was cached and inserted only to the master database. The failover connection would be set to *standard* mode, so the primary connection would be used when possible.

In a more complete cluster environment, where writes to either node would be replicated, a *sticky* failover connection could be used with *full history mode.*

Pertinent Settings

When working with various database architectures, there are a few settings in various parts of the system that are important.

Database connection settings - Failover Datasource

Any database connection can have a failover datasource. If the main connection is unavailable, any queries executed on it will pass through to the secondary connection. In this way, a secondary database can be used when the first is not available, and the system will continue to function. It is important to note that everything passed through to the failover will function normally- no special considerations will be made. For example, the system won't cache data for the primary connection, it will forward it to the secondary. In cases where you want to allow reading from the secondary database, but not writing, you can set up another connection directly to the first database, with no failover, and set all of your write operations to use that.

Clustering settings - History Mode

The history mode dictates how history will be treated when the node is not active. If *partial*, the data will be cached, and only forwarded when the master node is available. This mode can be used to prevent data from being inserted into a backup database in some cases.

3.12.6 Troubleshooting Redundancy

Troubleshooting Connectivity

When the two redundant nodes are connected, the will both appear along with their state details in the Status section of the gateway. There are also various other places where the redundancy state is shown as "connected". If the two nodes cannot connect, check the following:

© 2011 Inductive Automation

Verify that the master address is correct in the backup. Try to ping the master machine from the backup machine, and verify that you're using the correct address for the network card that the master is connected through.
If using system names (or domain names), verify that the name is resolving to the correct address by performing a ping.
Verify that the firewall on the master is set to allow TCP traffic to the designated port.
Verify that the backup is not connecting and then immediately disconnected for some reason. Viewing the error log in the gateway console section should show this. If errors are occurring at regular intervals, look at the message for an indication of what is happening. An example of a potential problem is when the failover time is set too low for the given network, which results in many socket read timeout exceptions, which in turn leads to many disconnect/reconnect attempts. If errors are occurring, but the cause isn't clear, contact Inductive Automation support.

Advanced Troubleshooting

A variety of loggers can be found under the gateway console section by going to "Levels" and searching for "Redundancy". By setting these loggers to a finer level, more information will be logged to the console. This is generally only useful under the guidance of Inductive Automation support personnel, though more advanced users may find the additional logged information helpful.

© 2011 Inductive Automation

Project Design

Part IV

inductive automation
800.266.7798
www.inductiveautomation.com

4 Project Design

4.1 Designer Introduction

The Ignition Designer is where the majority of configuration and design work is done in Ignition. It is used to configure *Projects*, which are the major unit of design. Projects contain various *resources*, such as *windows* and *transaction groups*. A project may contain a variety of different types of resources, depending on the goal of the project and the modules installed.

Common First Steps
Create some SQLTags
Create a Window
Create a Transaction Group

See also:
Launching the Designer
What is a Project?

4.2 Using the Designer

4.2.1 Logging into the Designer

Click on the "Launch Designer" button near the top-right corner of any Gateway page to launch the Ignition Designer. To log into the Designer, you need to use a username and password that:
1. Is valid for the *System Authentication Profile*. This is set in the Gateway Settings section of the Gateway's web configuration interface.
2. Has at least one of the roles defined in the *Designer Role(s)* field in the Gateway Settings page.

4.2.2 Creating or Opening a Project

After logging into the Designer, or at anytime through the Designer's `File > Open` menu, you will be prompted to either open an existing project or create a new project. A project must have a name and an authentication profile. You may also specify a default database connection and a default SQLTags provider for your project.

See also:
Project General Properties

4.2.3 Designer UI Overview

The Designer is organized around a central *workspace*. The workspace changes based on the type of *resource* that you are currently editing. For example, if you are editing a Window, then your workspace will be the Window Designer. If you are editing a Transaction Group, your workspace will be the Transaction Group Editor, etc.

There are many *dockable panels* that surround the workspace, as well as the familiar menu bars and toolbars. The dockable panels may be rearranged to your liking. Each type of workspace may have panels that are only valid when that workspace is active. Each workspace remembers its own *perspective*, which is the docking arrangement of the panels around it. If you have closed a panel and want to get it back, re-enable it in the `View > Panels` submenu.

© 2011 Inductive Automation

4.2.4 Using the Docking System

The Designer's docking system allows for a very flexible user interface, allowing a user to customize the layout to their liking. To re-arrange the dockable panels, simply drag on their title bars. As you are dragging the panel, use the highlighted border that appears to gauge where the panel will be moved to. Dockable panels can be in one of four modes:

1. Docked. The panel is visible, and located somewhere around the perimeter of the workspace. If two panels are docked in the same location, a tab strip will appear to switch between the two panels.
2. Floating. A panel can be dragged outside of the workspace perimeter to be floated. The panel can now be positioned anywhere on your desktop.
3. Pinned. Pinning a panel makes it minimize to one of the four sides of the Designer, represented by a small tab. Hover over the tab to use the panel.
4. Hidden. A hidden panel is not shown. You can open it again by selecting it in the View > Panels menu.

Toolbars can also be rearranged and floated to your liking. Simply drag on the "textured" left edge of the toolbar.

If you have re-arranged your panels into a layout that you don't like, you can quickly revert back to the default by selecting the View > Reset Panels option from the menu bar.

☞ *Expert tip:* Your docking preferences are stored under `%USER_HOME%/.ignition/*.layout`. If you really want to reset your preferences, remove these files and restart the Designer.

4.2.5 Communication Modes

The Designer has three communication modes that affect data flow to and from the Gateway:

Off: All database query traffic and tag subscriptions and writes will be blocked.
Read-Only: tag subscriptions and SELECT queries will work, but tag writes and UPDATE/INSERT/DELETE queries will be blocked.
Read/Write: All data will be passed through to the Gateway.

The mode can be switched at any time via the tri-state toggle selection in the main toolbar, or the radio buttons in the Project menu. The Designer starts up in **Read-Only** mode as a safety mechanism, so that you don't inadvertently write to a tag as you are designing. You can customize the designer's startup mode, see the Designer General Properties section.

A common beginner mistake is to forget to switch the mode to **Read/Write** when attempting to test a window's functionality in preview mode.

A component with the GW_COMM_OFF quality overlay

Experts often use the **Off** mode while designing a window to temporarily shut off data flow so that they can manipulate components' bound properties without the values being overwritten by the data bindings. This is useful to set the values that they want to serialize into the window. This can be important for windows with large datasets; clearing the datasets before saving the window can significantly reduce the size of the window, improving performance.

Note: This setting does not affect the execution of a project's *transaction groups*. This is because

© 2011 Inductive Automation

transaction groups execute on the Gateway, not in the Designer.

4.2.6 Designer Tools

4.2.6.1 Output Console

The *Output Console* is the script-writers best friend. It is a dockable panel, and can be opened via the `Tools > Console` menu or the `Ctrl-Shift-C` keyboard shortcut.

The output console is most frequently used to test and debug Python scripts in Ignition. By using the `print` keyword in your script, you can observe the inner workings of your script as it executes. For example, if you executed the following script:

```
# A function that intercepts tag writes, printing out the previous value first
def writeToTag(path, value):
   import system
   prevValue = system.tag.getTagValue(path)
   print "Writing value '%s' to %s, was previously '%s'" % (value, path, prevValue)
   system.tag.writeToTag(path, value)

writeToTag("Compressor/HOA", 2)
writeToTag("Compressor/HOA", 1)
```

It would print the following to the console:

```
Writing value '2' to Compressor/HOA, was previously '0'
Writing value '1' to Compressor/HOA, was previously '2'
```

Note that the output console is also available in the Vision Client, via the Diagnostics window.

See also:
About Python
Diagnostics Window

4.2.6.2 Diagnostics Window

The Diagnostics window, which is available in both the Designer and the Vision Client, contains a number of useful troubleshooting features. It features a number of tabs, some of which are initially hidden. Right-click on any of the visible tabs to show or hide other tabs.

Performance
Displays a number of small realtime charts that display various aspects of the currently executing Designer or Client's performance. These charts can be very useful to help troubleshoot performance issues, especially slow queries. One of the most common causes of query slowdown is simply running too many queries too frequently, and the *# of Select Queries / Second* chart can help identify when this is occurring.

Console
Displays the Output Console.

Log Viewer
Displays the logged events for the current Designer or Client session. Whenever errors occur, they will be logged and displayed in this tab. This is a good place to go when troubleshooting an issue, as any errors shown here may illuminate the cause of the problem. To view entries across all categories chronologically, uncheck the *Group Categories* checkbox.

Logging Levels

© 2011 Inductive Automation

Determines the verbosity of a host of internal loggers. Most users will not use this tab unless prompted by a technical support representative.

Thread Viewer
Shows information about the currently running threads. Most users will not use this tab unless prompted by a technical support representative.

4.2.6.3 Find / Replace

The Find / Replace tool is a very handy tool. It can be used to search an entire project for where a tag gets used. The replace feature can also be used to to make mass changes to a project with very little effort. To open the Find/Replace dialog box, choose the menu item under the Edit menu or use the shortcut Ctrl-F.

Finding

To search through your project, simply type what you're searching for in the text field at the top and press the Find button. You can use the wildcard character (*) which will match anything, and the single-character wildcard character (?).

For example, to find all references to a tag that include the string "`Motor`", you'd search for "`Motor*`". This would match things like "`Motor15`", "`MotorHOA`", etc, whereas the search query "`Valve?Status`" would match "`Valve1Status`" but not "`Valve38Status`"

Target Scope

To narrow down your search, it is often useful to specify a narrow search target. The Find / Replace system searches through many different parts of a project, and through SQLTags as well. The target settings let you specify exactly what to search through. By unchecking boxes in the target section, you can avoid search results that you aren't interested in.

Results

When you execute a search, all matching items appear in the search results section. You can double-click on an item in the results table to bring that item into editing focus in the Designer.

Replace

To use the replace feature, select a result entry after doing a search. You'll see the current value with the matching area in bold-face font. Enter the text you'd like to use as a replacement in the Replace text-box, and you'll be shown a preview of the new value in the preview box. Hit the Replace button to execute the replace. This will move your selection down in the results table so that you can rapidly execute multiple replacements. If you're satisfied that you'd like to make the identical replacement to many items, select them all in the results table in hit the Replace All button.

4.2.6.4 Image Manager

The Image Manager is available from the `Tools > Image Management` menu. This tool is a drag-and-drop browser that helps manage the images that are stored on the Gateway. It is important to realize that these images are shared across all projects: they are not stored inside a project itself.

Use the toolbar at the top to do common tasks like uploading new images and creating folders. You can drag images from your computer's desktop or hard drive into this window to easily upload new images to

© 2011 Inductive Automation

the Gateway.

You can also get to this tool by putting an *Image* component on a window, and using the browse button on the image's *Image Path* property.

See also:
Image Component

4.2.6.5 Query Browser

The Query Browser is a very convenient tool that lets you interact with all of the databases that you have configured connections for. Because Ignition is so heavily integrated with databases, it is very common in the course of project design to need to inspect the database directly, or to experiment with a SQL query to get it just right.

You can use the auto-refresh option in the Query Browser to monitor a database table for changes. This is often convenient when designing Transaction Groups. As the group runs, you can view the table that it is targeting with auto-refresh turned on to watch how the group is altering the table.

The Query Browser is a convenient way to make simple edits in a database table as well. If you execute a SELECT query that includes the table's *primary key(s)*, then you may activate e*dit mode* by selecting the *Edit* button. While in edit mode, you can alter the values in the result set. Make sure to hit *Apply* when you are done to commit your edits, or press *Discard* to back out. Note that this feature depends on the applicable JDBC driver's ability to detect the table's primary keys.

See also:
Creating a Database Connection

4.3 SQLTags

4.3.1 What is a SQLTag?

A SQLTag, in many ways, is what is simply considered a "tag" in other systems. They are points of data, and may have static values or dynamic values that come from an OPC address, an expression, or a SQL query. They also offer scaling, alarming, and meta information facilities.

SQLTags provide a consistent data model throughout Ignition, and offer the easiest way to get up and running creating status, control, and simple history systems. Despite their low initial learning curve, however, SQLTags offer a great amount of power in system design and configuration. The ability to aggregate tags from a variety of installations in a central SQL database means that you can build widely distributed SCADA systems more easily than ever before, with a high level of performance and relatively easy configuration.

For more information about the benefits of SQLTags, see the SQLTags Overview in the Architecture chapter.

Tag Execution

SQLTags are executed by *scan classes* inside of a *tag provider*. In a typical system there will be one or two tag providers (the internal provider, which keeps the tag configuration in the project, and possibly an external tag provider in which tag configuration and values are stored in a database), and a number of scan classes.

© 2011 Inductive Automation

SQLTags stored in an external provider will be available to all Ignition installations that have access to that database. One of the installations will be specified as the tag's *driver*. The driving system will have a copy of the scan class that it executes, which in turn evaluates the tag. The value will be stored to the database, and all of the other installations will be notified of the new value.

4.3.2 Types of SQLTags

There are several types of SQLTags that fall into three main categories, and represent six different types of execution.

Gateway Executed Tags

Tags executed in the gateway support all of the primary features of SQLTags: scaling, alerting, history, and role based permissions. They are identical in their configurations, apart from defining how the value is generated.

OPC Tags
OPC tags specify an OPC server and address which drives their values. The OPC address will be subscribed at the rate of the tag's scan class.

DB Tags
DB tags have three modes:

Static value - the tag has a value, and only changes when someone writes to it.

Expression - the tag value is generated by an expression. The expression syntax is the same as for property bindings, and allows mathematical operations, references to other tags, logic operations and more.

SQL Query - the tag will execute a SQL query each time it's evaluated (when its scan class runs) and will use the result as its value. Like SQL property binding, the queries may incorporate dynamic references to other tags.

System Tags

System tags, unlike gateway tags, are only available for use in the client. They are provided by the system, and therefore cannot be modified. They provide a variety of useful information about the system and the client status.

Client Tags

Client tags, as the name implies, are only available for use in clients. This means that their values are isolated to a client runtime, and even though they are created in the designer, each client will create their own instances. This makes them very useful as in-project variables, for passing information between screens, and between other parts of the clients, such as scripting.

While client tags are essentially Expression tags in that they can be static, expressions, or SQL queries, they do not have a scan class.

4.3.3 Creating SQLTags

Creating From OPC Tags

The easiest and most common way to create SQLTags is to drag tags into the *SQLTags Browser*

© 2011 Inductive Automation

window from the OPC Browser . After browsing OPC and finding the tags that you want, simply drag and drop them onto the correct tag provider, and the system will create OPC SQLTags for each.

Creating Tags Manually

Obviously the above method only works for OPC tags, and then only for browsable tags. For Expression and Static tags, as well as OPC tags that cannot be obtained through browsing, you can click on the "new tag" button or right-click on the provider node and select "New Tag".

Re-naming SQLTags

You can re-name a tag if you prefer to see something more meaningful in the *SQLTags* Browser. To change the name, select the tag you want to modify and right-click it to select 'Edit Tag(s)'. Select the ' General' tab and modify the 'Name' property.

Valid characters for *SQLTag* names include **spaces** and the following:
1234567890_-abcdefghijklmnopqrstuwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

4.3.4 Tag Properties

4.3.4.1 General Properties

Properties common to most tags

Property	Binding Name	Description
Name	`Name`	How the tag will be presented and referenced in the system. The tag path will be the provider, the folder structure, and this name.
Value	`Value`	The value of the tag. Can only be modified if the tag allows value writing and the user has sufficient privileges.
Quality	`Quality`	The data quality of the value. If not *GOOD (*integer value 192), the value should not be trusted. The Data Quality section explains quality codes in more depth.
Datatype	`Datatype`	The type of the value. It is important that this be set as correctly as possible with regards to the tag's underlying data source. The SQLTags system will attempt to coerce any raw incoming value (for example, from OPC or a SQL query) into the desired type.
Enabled	`Enabled`	Whether the tag will be evaluated by the scan class. If false, the tag will still be present, but will have a bad quality.
Access Mode	`AccessRights`	Specifies the access level allowed to the tag- read/write, read only, or custom. If custom, the tag will use the permission settings
Scan Class	`ScanClass`	The scan class that will execute the tag. The scan class dictates the rate and conditions on which the tag will be evaluated.

Additional properties - OPC Tags

Property	Binding Name	Description
OPC Server	`OPCServer`	The server against which to subscribe the data point.
OPC Item Path	`OPCItemPath`	The path to subscribe to on the server. The point will be subscribed at the rate dictated by the scan class.

© 2011 Inductive Automation

Additional properties - Tag in external providers

Property	Binding Name	Description
Driver	`DriverName`	The name of the Ignition gateway that will be responsible for the execution of the tag. All other gateways will monitor the value.

4.3.4.2 Numeric Properties

The numerical properties are available to OPC, DB, and Client tags whose data types are numeric.

Property	Binding Name	Description
Scale mode	`ScaleMode`	If and how the tag value will be scaled between the source, and what is reported for the tag.
Deadband	`Deadband`	A floating point value used to prevent unnecessary updates for tags whose values "float" by small amounts.

Scaling Settings

Property	Binding Name	Description
Raw Lo	`RawLow`	Start of the "raw" value range
Raw Hi	`RawHigh`	End of the "raw" value range
Scaled Lo	`ScaledLow`	Start of "scaled" value range. Raw low will map to Scaled low for the tag.
Scaled Hi	`ScaledHigh`	End of "scaled" value range. Raw high will map to Scaled high for the tag.
Clamp Mode	`ClampMode`	How values that fall outside of the ranges will be treated. "Clamped" values will be adjusted to the low/high scaled value as appropriate.

Linear Scaling

The value will be scaled linearly between the low and high values, and clamped as appropriate.

The linear equation is:

$$ScaledValue = \Delta S * (Value-R_L)/\Delta R + S_L$$

Square root Scaling

The equation for square root scaling is:

$$ScaledValue = (\Delta S * (Value-R_L)/\Delta R) + S_L$$

... where ΔS is (ScaledHigh-ScaledLow), ΔR is (RawHigh - RawLow), R_L is RawLow, and S_L is ScaledLow

4.3.4.3 Metadata Properties

The metadata properties provide informational properties for a tag. The values of these fields can be read and modified through scripting, or bound to properties such as range, tooltips, etc.

Property	Binding Name	Description
Format	`FormatString`	How the value should be formatted when converted to a string (only applies to numerical data types)
Eng. Units	`EngUnit`	The engineering units of the value
Eng. Low	`EngLow`	The lowest expected value of the tag.
Eng. High	`EngHigh`	The highest expected value of the tag
Tooltip	`Tooltip`	The tooltip provides a hint to visual components as to what should be

© 2011 Inductive Automation

		displayed when the user hovers their mouse cursor over the component that is being driven by the value of this tag.
Documentation	`Documentation`	A freeform text property for information about the tag

4.3.4.4 Permission Properties

By default, a tag's Access Mode property is set to **Read/Write**, which means that any user may read the value of the tag and may write to the tag. **Read-only** mode makes the tag non-writeable for all users.

Custom mode allows the tag to assign read/write or read-only privileges to *individual roles*. Any roles not explicitly granted a right by using the custom permissions editor will not be able to read the tag's value or write to the tag.

4.3.4.5 History Properties

The properties on the *History* tab detail if and how the tag's history will be stored in the SQLTags Historian system.

Property	Binding Name	Description
Store History	`HistoryEnabled`	Whether the tag will report its history to the SQLTags Historian system.
History Provider	`PrimaryHistoryProvider`	Which SQLTags Historian data store the tag will target. A particular tag can only target one history store.
Historical Scan Class	`HistoricalScanclass`	The scan class to use to evaluate tag history. This allows the tag's history to be stored at a slower rate than the status is updated at.
Historical Deadband	`HistoricalDeadband`	A deadband that applies only to historical evaluation.
Value Mode	`InterpolationMode`	How interpolation will be handled for the tag in querying. See below for more information.
Max Time Between Records	`HistoryMaxAgeMode / HistoryMaxAge`	The maximum amount of time that can pass before a new record is logged for the tag.
Timestamp Source	`HistoryTimestampSource`	Which timestamp is used for the value of the tag.

Value Mode

The value mode, *analog* or *discrete*, dictates the type of value that the tag represents, and will be used in querying to determine how values should be interpolated. *Interpolation* is the method in which the SQLTags Historian query system generates values for a tag when the desired time does not fall directly on a sample timestamp.

Discrete
The value will not be interpolated. The value returned will be the previous known value, up until the point at which the next value was recorded.

Analog
The value will be interpolated linearly between the last value and the next value. For example, if the value at $Time_0$ was 1, and the value at $Time_2$ is 3, selecting $Time_1$ will return 2.

Max Time Between Records

Normally SQLTags Historian only stores records when values change. By default, an "unlimited" amount

© 2011 Inductive Automation

of time can pass between records- if the value doesn't change, a new row is never inserted in the database. By modifying this setting, it is possible to specify the maximum number of scan class execution cycles that can occur before a value is recorded. Setting the value to 1, for example, would cause the tag value to be inserted *each execution*, even if it has not changed. Given the amount of extra data in the database that this would lead to, it's important to only change this property when necessary.

Timestamp Source

When a SQLTag executes, there are two possible timestamps that can be observed: the time associated with the data, and the time that the tag was evaluated. The first case is generally only interesting when the value is provided by an OPC server. In most cases, the time provided by OPC, which in Ignition is referred to as the "Value" time, will be very close to the system time. Some servers, however, either due to their location or how they function (history playback, for example), will provide times that are very different than the current time.

It is generally desirable to store the System time, as it is the time that the value was actually observed by the system, and it creates a uniform timeframe for all realtime data. However, in the later case described above, it is necessary to store the time provided by the OPC server. Using the Value timestamp source has several consequences: the system is no longer able to validate the tag quality against the scan class' execution, and tag value interpolation will behave differently.

The validation of the scan class execution is generally not a concern when recording historical playback data. Interpolation only occurs when the value mode is Analog, and when there is not a value for every time window. Using System time, the value is only interpolated during the last "scan class execution window", that is, one scan class timeframe before the next value. Using Value time, however, the value is interpolated for the entire time between two data points.

4.3.4.6 Alerting Properties

SQLTags have the ability to define both *digital* and *analog* alerts- conditions of particular interest that can be used to generate emails, store records in the database, and more.

Digital Alerts

Digital alerts define a specific value that represents the "active" state, as opposed to Analog alerts, which define a range.

Alert Name
The name of the digital "state". Will be shown in the alert log and status systems.

Severity
The relative "importance" of the alert. Can be used for filtering purposes later.

Value Mode - Equal/Not equal
Alert is active when the tag's value matches the specified value.

Value Mode - Any change
Alert occurs any time the tag's value changes, subject to the alert deadband. "Any Change" alerts are instantly clear, as well, as there is no defined clear state.

Time Deadband
The alert is only considered active once the "active state" has been true for the given amount of time. If the state changes before the time deadband clears, no alert is generated.

© 2011 Inductive Automation

Analog Alerts

Analog alerts define any number of "states" - each of which defines a range, severity and name. The settings for a state are similar to those for a digital alert, with a few differences:

Low and High Setpoints
Define the range in which the alert state is considered "active". Outside of the range the state is "clear". May be "infinite" in order to have unbounded state ranges. For example, an alert state range with a lower bound of 50.0 and an upper bound of infinite will be active for any value greater than 50.0.

Setpoint Mode
Dictates how the state acts when the value is on the boundary of the state. "Inclusive" means the setpoint is included in the range of possible values, and the state will be active if the tag's value equals the setpoint value. "Exclusive" excludes the setpoint value from the range.

Tag Driven
Both the low and high setpoint values can be driven by a separate tag. The values of the referenced tags will be latched each time the state is evaluated, and will otherwise act like static values.

Alert on any change
An alert will be generated for any value change while the value is inside the boundaries of the state.

General Settings

Ack Mode
Dictates how acknowledgement works for the alarm.

Unused - Acknowledgement will not be used for this tag, and any alert that is generated will automatically be marked as acknowledged.

Auto - The alert is acknowledged automatically when the alert state becomes cleared.

Manual - The alert is never set to acknowledged by the system, and it is up to the user to manually acknowledge alerts.

Timestamp Source
Specifies which timestamp should be reported for the active/clear times- the time coming from the system, or the time coming from the tag value.

System - The timestamp will be the current system time when the alert event occurs.

Value - The timestamp used will be the timestamp associated with the value that caused the event.

Alert Deadband
Defines a deadband that is only used when evaluating the alerts. This setting is used primarily with analog alerts to prevent many alerts from occurring for analog values that constantly "float".

An alert with a deadband will become active immediately after the tag's value crosses the active threshold. The tag will not clear, however, until after the alert has gone outside of the active range by more than the deadband. In most cases, the deadband is added or subtracted to/from the setpoint to determine clear. In *any change* mode, the tag will only generate a new alert when the value has changed by more than the deadband from the last alerted value.

Time Deadband
Defines an amount of time that the tag value must remain in the numeric region considered "active" before the alert is considered active. Once the alert has become active (after the time deadband

© 2011 Inductive Automation

specified has elapsed and the value is still in active range), the alert will clear as soon as the value leaves the active region.

For example, suppose you had a digital alert that became active when the tag value is 5 with a 1 minute time deadband. Suppose the tag's value becomes 5 at 3:15 pm. The tag's alert will only be considered active at 3:16 pm, as long as the value remained 5 that entire time.

Display Path

This is an arbitrary path that can be used for querying and display purposes later. For example, if this path is not empty, it will be used by default to identify the alert by the Vision module's built-in alert status table instead of the path to the tag itself.

Notes

Freeform text field that can be used to record information about the alert. Can be used for display purposes later.

Notification Settings

These settings are used for sending email alerts in association with Alert Notification Profiles that are configured in the Gateway.

Send Clear

Indicates that a message should be send when the alert clears, in addition to when it becomes active.

Message Mode

How the message should be generated for the alert.

Auto Generated - The system will create a basic message describing the alert condition.

Custom - The provided message will be used.

Custom Subject

The subject of the email that will be sent for the alert. Can include references to other tags and alert properties, as outlined below for the message.

Custom Message

The message to be sent for the alert.

Custom messages can reference other tags, and several properties of the alert. The following alert properties may be referenced:

TIME

VALUE

STATE_NAME

ALARM_FLAGS - Numeric representation of the current message. Can be a combination of the following:

0x1 - Register - Indicates that the tag has just been loaded and is being registered with the system.

0x2 - Active - The alert is active

0x4 - Cleared - The alert is clear

0x8 - Acknowledged - The alert has been acknowledged

0x10 - Deregister - The alert is being de-registered, likely due to tag deletion.

ALARM_TYPE - User friendly alert state message, either "active", "clear", or "acknowledged".

ITEM_PATH

SEVERITY

© 2011 Inductive Automation

DISPLAY_PATH

NOTES

SYSTEM

To reference a property, put the name inside of square brackets, inside of the curly braces normally used for references. For example, `{[ALARM_TYPE]}`

To reference a tag, use the standard curly brace syntax. For example, `{North Area/Compressor1/State}`. The path may be wrapped in square brackets as well, if using formatting as described below.

Referenced value number formatting

Values referenced in the subject or message can be formatted for display. To do this, the variable name or tag path must be followed by a pipe "|" symbol and a format pattern. The format pattern can be for a date or a number (as described in the documentation for dateFormat and numberFormat, respectively). In order to use formatting with tag references, tag paths must be enclosed by square brackets. For example, the following would display only two decimal places:

```
{[North Area/Compressor1/Amps|#0.00]}
```

To format a date tag:

```
{[North Area/Compressor1/LastChange|MMM d, yyyy]}
```

Default Messages

If not using custom messages, the default will have the following format:

Subject: {[ITEM_PATH]} {[ALARM_TYPE]}

Body: Alert {[ALARM_TYPE]} {[TIME]} - {[ITEM_PATH]} {[STATE_NAME]} {[VALUE]}

4.3.4.7 Expression/SQL Properties

DBTags have the ability to use an *expression* or a *SQL query* as their value instead of an OPC item path. This can be used to select information from the database or create your own formulas to manipulate other tag values .

Expression

In expression mode, the tag can use all of the features available in the *expression language*. It can refer to other tags, and use operators and functions to calculate a value for the tag.

See also:

Expressions Overview

SQL Query

In this mode, the tag's value will be the result of the specified SQL query. The query can be any valid query, but should result in only one value. Note that insert and update queries can be used, and will often result in an integer value, so the tag's data type should be set accordingly.

Like SQL Query bindings in the Vision module, the queries for tags can refer to other tag values. The values of referenced tags will inserted as literal text in the query before being sent to the database.

© 2011 Inductive Automation

4.3.5 Scan Classes

Scan classes dictate the execution of SQLTags, and therefore play a crucial role in the design of large, high-performance systems. They offer several key modes to help create efficient projects.

Creating and Editing Scan Classes

Scan classes are created by clicking on the *Edit Scan classes* button in the SQLTags browser toolbar. The window will appear with a list of configured scan classes on the left, and configuration settings on the right.

Scan Class Properties

Scan Class Name	Unique name of the scan class.
Mode	Described below
Slow Rate	Base update rate, specified in milliseconds, at which tags will be executed.
Fast Rate	Used by the *Driven* and *Leased* modes, this is the faster rate that the tags will be executed at when those modes are active.
Stale Timeout	How long to wait before the tags in the scan class are determined to be "stale" (not running). This is calculated off of the last expected execution time of the scan class, and is particularly important for scan classes executed by other drivers through the external SQLTags provider.
Driven Properties	Used by the *driven* mode to determine when the scan class should run at the fast rate.

Note on rates: If the rate is set to 0, the scan class will not be executed. It is common for leased and driven modes to use 0 as a slow rate in order to achieve an "on/off" effect.

Scan Class Modes

Direct

The scan class executes at a fixed rate, defined by the *slow rate* setting.

Leased

The scan class executes at the fast rate when any of the tags it contains are subscribed and visible in a client window. If no tags are subscribed, the scan class runs at the slow rate.

Driven

The rate of the scan class is based on the value of a driving tag. The condition is a simple comparison between an a tag value and a number. If the condition is true, the scan class will execute at the fast rate.

It's useful to keep in mind that the driving tag can be an Expression tag that performs complex calculations and references other tags. In this way, it's possible to create robust scan class triggering.

Historical Scan Classes

Historical scan classes are simply standard scan classes used by SQLTags to store history. By utilizing separate scan classes for status and history, it's possible to maintain a tag's status at a fast rate, without storing large amounts of history unnecessarily.

Despite the fact that there is not a technical differentiation between standard and historical scan classes, it is recommended that you create separate scan classes for each purpose and name them in

© 2011 Inductive Automation

a manner that indicates their usage. It is common to modify scan classes in order to affect a large number of tags, and without a consistent distinction it may be possible to affect tag execution in unexpected ways.

4.3.6 Tag Paths

Tags and their properties can be referenced by a string based path. Each has a unique absolute path, and will often have many equivalent relative paths when referenced from other tags. You will most often generate these by browsing or through drag and drop. However, it's a good idea to understand how tag paths work, particularly if you get into indirect tag binding or scripting.

A tag path will look something like this: `[Source]`*`folder/path/tag.property`*

The italicized portion of the path may contain the following:
A tag
Any number of nested folders followed by a tag, separated by forward slashes (/).
A period (.) followed by a *property name* after the tag. Omitting this is equivalent to using the .Value property.

Now consider the **[Source]** (portion surrounded by square braces)

Source Option	Meaning	Applicability
`[Tag Provider Name]`	The name of the tag provider that hosts the tag	OPC and Expression tags
`[]` or not specified	The default tag provider for the current project.	OPC, Expression tags
`[.]`	Relative to the folder of the tag that is being bound.	Expression, Client tags
`[~]`	Relative to the tag provider of the tag that is being bound (root node)	Expression, Client tags
`[Client]`	Refers to a client tag	Client
`[System]`	Refers to a system tag	System

Relative Paths

Paths that begin with [.] or [~] are known as *relative paths.* The are used inside SQLTags that bind to other tags, and are relative to the host tag's path. Using the relative path syntax helps avoid problems cause by moving tags and renaming providers. `[~]` refers to the tag's provider root. It can replace the explicit provider name, and thus protect against provider renames and importing/exporting/moving tags between different providers. `[.]` refers to the tag's current folder. By using `[.]`, tags can be moved from folder to folder without problem (provided that all of the applicable tags are moved together).

4.3.7 Data Quality

Data Quality is the measure of how reliable a particular SQLTag's data is. If a tag's quality is not *Good*, the value generally should not be trusted. There are a wide variety of causes of bad data, from network disconnections to software failure, to invalid tag configuration. The quality is a property of the tag (`Quality`), and can be seen in the SQLTags browser. Additionally, bad tag qualities will be reflected in components bound to tags through the *quality overlay* system.

The following table outlines the primary data qualities. There are more values, but these represent the most common:

© 2011 Inductive Automation

Quality	Meaning
Good	The data has met all criteria for being considered reliable.
Bad	The data is not reliable, further data isn't available.
Stale	The tag has not been evaluated within the expected time frame. There is likely a deeper problem with the tag provider.
Config_Error	There is a problem with the tag's configuration. The error log may provide more information as to the exact problem.
Comm_Error	There is a problem in communication somewhere between the tag and its data source.
Tag_Exec_Error	There was an error evaluating the tag.
Expression_Eval_Error	The expression in the tag generated an error during execution. The error log should provide more information on the error.
Type_Conversion_Error	The value of the tag could not be converted to the requested data type. Check the assigned data type of the tag.
OPC_Not_Connected	The OPC server driving the tag is not currently connected OR a value has not yet been received by the tag from the server.
Not_Found	The tag, or a tag referenced from inside of it, could not be found (incorrect reference path).
Driver_Demo_Timeout	The system driving the tag is operating in demo mode and has timed out.
GW_Comm_Off	When viewing SQLTags in the designer, the tags will have this value if communication with the gateway is turned off from the toolbar.
Access_Denied	The tag permission settings do not allow the current user to view the tag.
Disabled	The tag's "enabled" property has been set to *false*.

More information about Quality Overlays.

Tag Quality and Referenced Tags

When tags reference other tags, such as in expressions, they will often pass the worst sub-quality up as their own. For example, even though a particular tag's expression executes without problem, if the expression references a tag whose quality is "Bad", the expression tag will also report "Bad".

4.3.8 Importing/Exporting using CSV

It is possible to export and import SQLTags to/from a CSV file format using the toolbar on the SQLTags browser window. Simply click the Export button to export, or Import to load a previously exported file.

4.4 Project Properties

4.4.1 Project General Properties

A project's general properties apply to the project as a whole, across all module functionality. You can edit a project's general properties in the Designer by double-clicking on the `Configuration > Properties` node in the Project Browser, or by navigating to the `Project > Properties` menu.

Note that a few properties of a project, such as its name, description, and title are set in the Gateway by clicking on the *edit* link next to a project under the `Configuration > Projects` section.

Important Concept: Defaults
Project General Properties is where you set the project's *Default Database* and its *Default SQLTags*

© 2011 Inductive Automation

Provider. It is important to understand how to use defaults effectively for proper project design. Wherever you use a database connection or a SQLTag in a project, you are always given the option to use the project's default, or an explicitly named connection or provider. If your project is like most typical projects, it primarily uses a single database and a single SQLTags provider. By consistently using the "default" option, you make your project more resilient to change.

For example, suppose you have designed a project, and it has a database connection called "Production_DB". Now you want to adapt the project to a new, similar plant, while leaving the existing project intact. You copy the project and create a new database connection, called "New_DB". If your project consistently used it's default database connection, the switchover will be as simple as changing the copied project's default database. However, if you used the explicit "Production_DB" connection in your groups and screens, you will need to laboriously switch the bindings over to "New_DB".

SQLTags Settings
The SQLTags provider chosen here will act as the project's default provider. To use the default provider, simply omit the *source* section of a tag path, or leave it blank, for example: `Path/To/MyTag` or `[]Path/To/MyTag`. The client poll rate is the rate at which a Vision Client or Ignition Designer polls the Gateway for updates to its subscribed SQLTags.

Database Settings
The default database connection to use for this project. To use the default database connection, use the special <*default*> connection, or in scripting, the empty-string connection `""`.

Security Settings
Choose the authentication profile that governs this project's security. This profile will be used for client logins. You may also optionally specify a list of roles that are required for a user to log into this project. Use commas to separate the roles. Users must have *all* of the roles in order to log in. If no roles are specifed, the user only needs to correctly authenticate with the authentication profile in order to log in.

Auditing Settings
If auditing is enabled, audit events will be stored that relate to this project in the chosen audit profile.

Publishing Settings
This is where you configure whether or not a project is split into separate staging and published versions. By choosing "Manual" publish mode, pressing Save in the the Designer will alter the Staging version of the project. The Published version of the project will only be updated when you hit the "Publish" button. If you are in "Auto" publish mode, each save acts like a save followed by a publish, so the two versions are always the same. You can also specify here whether or not commit messages are required, and if so, under what conditions.

See also:
Project Management
Tag Paths
Security Overview
Project Versioning

4.4.2 Designer General Properties

These properties are used to configure how the designer acts.

Startup Options
You may choose what Comm Mode the Designer starts up in. Learn more in the Communication Modes section.

© 2011 Inductive Automation

4.4.3 Designer Window Editing Properties

These options affect the operation of the Designer as it applies to the Vision module's window design.

Default Color Mapping
The color mapping defined here will be the initial color mapping when configuring a new number-to-color property binding.

Default Component Layout
The layout constraints specified here will be the layout constraints used for all newly added components. If you wanted to effectively "disable" relative layout, you would change this setting to Anchored with the North and West anchors selected. Learn more in the Component Layout section.

Component Manipulation
These options affect how the user interface to manipulate components acts. Altering the handle opacity can be helpful when dealing with lots of very small components, so that you can see through the resize handles to align the component perfectly.

Component Bounds
Disabling the constraint on parent bounds allows you to position components outside of their parents bounds, which can be helpful in advanced layouts.

Window Committing
By default, every time you close a window, you are prompted whether or not you wish to *commit* the window. Choosing "yes" will serialize the window and mark its project resource dirty, so that next time you save the project the window will be updated. Choosing "no" will effectively revert all changes to the last time the window was committed. This option allows you to skip the commit prompt, opting to always commit the window on close.

4.4.4 Client General Properties

These properties apply to the Vision Client in general.

Timezone Behavior
The Vision Client can emulate any timezone. By default, it will appear to be in the same timezone as the Gateway. This has the effect of all clients behaving the same, regardless of the timezone setting on the Client's host operating system. Depending on your project's requirements, this may not be optimal. You may have the Client use the host's timezone by choosing the "Client Timezone" option, or you may specify an explicit timezone for all Clients to emulate.

Publish Mode
This setting affects how clients receive updates when the project is saved. The default is *Notify*, which means that all running Clients will display a yellow information bar at the top of their display that notifies the operator that an update is available. The update will be installed when the operator clicks on the yellow bar. You may choose *Push* mode to have updates automatically pushed to all running clients with no operator interaction. This is often desirable when a client is running in a situation where keyword and mouse access is inconvenient, such as in a large overhead display.

Touch Screen
All clients can operate in touch-screen mode. When in this mode, clicking on numeric and text entry boxes will pop up on-screen keyboards that can be used for data entry. By enabling touch-screen mode, an operator is given the opportunity to activate the mode on the startup screen. You have the opportunity to control whether or not clients start up with touch-screen mode active by default or not as well. These settings are helpful for mixed-use projects, i.e. those that are launched on both touch-screen devices and traditional computers and laptops.

© 2011 Inductive Automation

4.4.5 Client Launching Properties

These properties apply to the Vision Client launch process.

Launch Icon
This image will be used to represent the project on the launch page and desktop shortcut. This needs to be a path to an image that has been uploaded to the Gateway. Use the browse button to choose or upload a new image.

Gateway Launch Page
The default launch mode determines what kind of launch occurs when the user hits the "Launch" button that appears next to the project in the Gateway home page. Each launch mode can also be enabled individually, which will turn the launch button into a split-button, allowing the user to choose the launch mode.

The project can also be hidden from the launch page, which is often useful for projects that are still under development. These projects can still be launched from the Designer's `Tools > Launch` menu.

Java Web Start Properties
These properties affect how the launched project will appear when launched through one of the Java Web Start launch modes: *WIndowed* or *Full Screen*. The *Vendor* and *Homepage* properties will be displayed in the Java Application Manager, which you can find through the Java Control Panel on a Windows computer.

The start maximized button will make the application start in a maximized window. Note that this is not the same thing as *full-screen mode*, which is only available when the client is launched in full-screen mode. In full-screen mode, the width, height, and start maximized properties have no effect.

When launched in full-screen mode, the user will be given an "Exit" button on the login screen by default. For terminals where the application should not be exited, this button can be removed by checking the "Hide Exit Button" checkbox.

Applet Properties
These properties affect how the project appears when launched as a browser applet.

Client Memory
These properties govern how the client uses RAM resources on its host machine. The initial memory setting is how much memory the client will require on startup. While this is typically left alone, boosting it a bit can improve performance somewhat.

The maximum memory setting sets a cap on how much memory the Java VM is allowed to use. This setting can be important for clients that require very large charts, tables and reports. Even if you have launched a client on a machine with plenty of RAM, you'll also need to boost this setting to allow the client to use more RAM.

See also:
Image Management
Launching Clients

4.4.6 Client Login Properties

These properties affect how the Vision Client's login process behaves and appears.

Login Screen
These properties affect the appearance of the login screen. By default, the title area of the login screen will contain the project's title (or its name, if the title is blank), along with the project's description. You can override this by entering a welcome message for your project here. You may

© 2011 Inductive Automation

use HTML to format the message. You can also set an image to use instead of the Ignition logo on the login screen's header. You may also override the text used in the login controls.

Auto Login
By enabling auto-login, you can have the launched client skip the login process. The client will log in behind the scenes using the credentials supplied here. If they fail, the login screen will be presented.

4.4.7 Client Polling Properties

This property affects how the client polls for information.

Important Concept: Polling Rates
Throughout the design of a Vision project, it will be common to have data bindings that run SQL queries. Those bindings all have the option to *poll*, or run repeatedly on a timer. By default, all bindings poll at a rate relative to the *Base Rate*. This is important, because it allows the designer to globally speed up or slow down the rate at which queries are run. This can be helpful when troubleshooting performance problems.

4.4.8 Client User Interface Properties

These properties affect how the Vision Client appears and behaves while it is running.

Minimum Size
Typically, a Vision Client is designed to run on multiple different resolution monitors. The various component layout features help design elastic screens, but sometimes you need to set a lower bound as to how small you'll allow the client's usable area to shrink. This is what the *Minimum Size* settings are for. You can see these settings visually represented in the Designer as lines on the Vision workspace. Whenever the usable space shrinks smaller than these bounds, scrollbars will appear, capping the width and height to these minimums. This defaults to 800x600.

Client Background Color
This option allows you to specify the color of the Vision workspace, which will be visible when not obscured by windows.

Client Menu
These options allow you to alter the appearance, or remove completely, the menu bar that appears in a running Vision Client.

See also:
Component Layout
Menu Bar Scripts

4.5 Project Scripting Configuration

4.5.1 Script Modules

A project's *Script Modules* are a global library of scripts that can be called from anywhere within the scope of a project. These scripts are organized as named modules that all live under the `app` module. To open the **Script Module Editor** double click on the `Configuration > Script Modules` node in the Project Browser or navigate to the `Project > Script Modules` menu.

Rule of Thumb: Never Copy-and-Paste a Script

If you're unsure of when to put scripts in a script module vs embedding the script directly in an event

© 2011 Inductive Automation

handler, follow this simple rule. If you ever find yourself copying a script from one event handler to another, *stop* and refactor the script into a global script module! Then simply call your new module from the event handler. This rule will help prevent code duplication across your project, a major maintenance liability.

How to use Script Modules

To add a script module, simply select the `app` package and press the *New Module* button. Each module is a python script that may define many functions. You may also organize modules in sub-packages if you'd like. Lets suppose you added a script module named `myfuncs`, whose body was:

```
def callMe(message):
    import system
    system.gui.messageBox(message)
```

Now, anywhere in your project you can call:

```
app.myfuncs.callMe('Hello World')
```

Whats up with that "`import system`" call?
Frequently in Ignition, your scripts get `system` (the built-in library package in Ignition) and `app` (your project's global script modules) imported for you automatically. Whenever you define a new scope (which you've done with `def`), we can no longer do this for you, and you'll need to import them manually.

See also:
About Python
Scope and Import

4.5.2 Event Scripts

4.5.2.1 Overview

Projects may use scripting to react to a variety of events and actions that occur within the project's lifecycle. There are two major scopes for scripting: Gateway scripts and Client scripts. Gateway scripts execute on the Ignition Gateway, which means that they always execute in one place. If you are running a cluster, then these scripts execute on the current Master node. Client scripts execute in the client, which means that they may never execute (if no clients are running), or they may execute many times. Client scripts will also execute in the Designer, but only in *Preview Mode*.

Note that these project global event scripts are not to be confused with the component event handler scripts.

4.5.2.2 Startup and Shutdown Scripts

These script types are available in both Gateway and Client scopes. These scripts will be run when the project starts up or shuts down.

In the Gateway scripting scope, this means that the script will run when the Gateway starts up or is shut down, *and* whenever the scripting configuration changes via a Designer save action. This means that while designing, the startup and shutdown events may happen frequently.

In the Client scripting scope, these scripts run after a user successfully logs in or out, or when the client is closed.

4.5.2.3 Shutdown Intercept Script

This script type is only available in the Client scope. This is a special script that will be called when the user tries to exit or close the client. This script is run with a special `event` variable in its namespace.

© 2011 Inductive Automation

When the script terminates, if `event.cancel` is `1`, then the shutdown will be aborted, and the client will remain open. Otherwise, the normal shutdown script will be called, and the client will close.

Example

```
if "SuperUser" not in system.security.getRoles():
   system.gui.warningBox("Exit not allowed for non-admin user.")
   event.cancel=1
```

4.5.2.4 Keystroke Scripts

Keystroke scripts are only available in the Client scope. These are scripts that run on a certain key combination. You may add as many keystroke scripts as you'd like, as long as each one has a unique key combination.

When choosing a keystroke, you may choose any number of modifiers, which are keys or mouse buttons that must be down to activate the keystroke. You can also choose whether or not the keystroke is on the *pressed* or *released* event of a keyboard key, or upon the typing of a character. Special keys like the F-keys, ESC, etc, are only available in the pressed and released actions.

4.5.2.5 Timer Scripts

Timer scripts are available in both Gateway and Client scopes. These scripts execute periodically on a fixed delay or rate. Remember that Client timer scripts may never execute (if no clients are open) or may execute many times (once per open client). If you need scripting logic that occurs centrally, make sure you use Gateway scoped scripts.

Fixed delay or fixed rate?
A fixed delay timer script (the default) waits for the given delay between each script invocation. This means that the script's rate will actually be the delay plus the amount of time it takes to execute the script. This is the safest option since it prevents a script from mistakenly running continuously because it takes longer to execute the script than the delay.

Fixed rate scripts *attempt* to run the script at a fixed rate relative to the first execution. If they script takes too long, or there is too much background process, this may not be possible. See the documentation for java.util.Timer.scheduleAtFixedRate() for more details.

Shared thread or dedicated thread?
All timer scripts *for a given project* that choose "Run in shared thread" will all execute in the same thread. This is usually desirable, to prevent creating lots of unnecessary threads. However, if your script takes a long time to run, it will block other timer tasks on the shared thread. The rule of thumb here is that quick-running tasks should run in the shared thread, and long-running tasks should get their own thread.

4.5.2.6 Tag Change Scripts

Tag Change scripts are available in both Gateway and Client scopes. Each tag change script can be given a list of tag paths. Whenever one of these tags changes, the tag change script will execute. They will also get an initial execution whenever the scripting system starts up.

Each tag change script can be given a name for organizational purposes. To specify multiple tag for a given script, enter them one per line in the tag paths text area. To quickly import many tags, you can drap-and-drop tags from the SQLTags Browser window onto this text area.

These scripts receive three special variables in their namespace when they are run: `event`, `initialChange` and `newValue`. The `intialChange` variable is a flag (0 or 1) that indicates whether or not this event is due to initially subscribing or not. The `event` variable is a `TagChangeEvent` object,

© 2011 Inductive Automation

which itself contains the properties: `tag`, `tagPath`, and `tagProperty`. The third, `newValue`, is the new value for the tag property that is subscribed. These values are objects themselves that contain a value, quality, and timestamp. The following example script should be a good starting point.

Example

```
print "Received tag change event for %s" % event.tagPath

value = newValue.value
quality = newValue.quality
timestamp = newValue.timestamp

print "value=%s, quality=%s, timestamp=%s" %(value, quality, timestamp)
```

Tip: The TagPath object that you access via `event.tagPath` is itself a complex object. You can turn it into a string if you want the whole tag path by using the `str()` function. You can also access individual parts of the tag path. The most useful is usually the `itemName` property, which is the name of the tag represented by the path. To get the name of the tag, you can use `event.tagPath.itemName` .

4.5.2.7 Menu Bar Scripts

The Client's menu bar is configured through the Client Event Scripts dialog box. Each node in the menu bar that does not have children executes a script when the user presses it. Most commonly, these scripts will execute navigation actions; opening or swapping a window.

See also:
Typical Navigation Strategy
Client User Interface Properties

4.6 Transaction Groups

4.6.1 Introduction

Transaction Groups are the heart of the SQL Bridge module. They are units of execution that perform a variety of actions, such as storing data historically, synchronizing database values to OPC, or loading recipe values. A variety of group types, items types, and options means that Transaction Groups can be configured to accomplish almost any task.

The Transaction Group Workspace

Transaction groups are edited through the Ignition designer. When a group is selected, you will be presented with the transaction group workspace.

The workspace is broken into several parts:

1) Title bar - Shows the name of the currently selected group, as well as options to set it as Enabled or Disable, and to Pause, if it's currently executing.
2) Item configuration - Shows all of the items configured in the selected group. Many settings can be modified directly through the display, the rest by double-clicking the item, or selecting "edit" in the context menu.
3) Action / Trigger / Options tabs - Define how and when a group executes. Holds most of the options that apply to the group in general, such as the update rate, and which data connection it uses.
4) Status / Events tabs - Provides information about the executing group, including the most recent

© 2011 Inductive Automation

messages that have been generated.

Enabling Group Execution

In order for groups to be evaluated, they must first be enabled. This is done by selecting "enabled" in the group title bar, and then saving the project. The group executing can be stopped by reversing the procedure and selecting "disabled" before saving. If you want to quickly and temporarily stop the group's evaluation, toggle the "pause" button. This will prevent execution until the group is un-paused, or until the system is restarted.

Editing Group Settings

Group settings may be modified at any time, regardless of whether or not the group is executing. Modifications will be applied when the project is saved, and the group will be started or stopped as required. Some changes such as modifying items may cause features like live values to appear to be incorrect. It is therefore important to note the modified icon that appears next to the group, and to save often. If you would prefer to stop the group before making edits you can simply pause the group. Execution will begin again after the project is saved.

4.6.2 Anatomy of a Group

4.6.2.1 Action Settings

The *action settings* of a group define how often the group will be evaluated, as well as important settings that apply to the group as a whole.

They are found on the tab labeled "Action", the first of the tabs on the right side of the Transaction Group workspace.

Common Settings

The settings vary for the different types of groups, but a few setting are common to most of them:

Update rate	How often the group is evaluated. For a number of reasons, the group may not execute during the evaluation. The most common reason is the trigger, but see Execution Cycle for more possible reasons why evaluation will exit.
Data source	The data connection to use for the group. Can be "Default", which will use the default connection for the project.
Update mode	For groups that support it, sets the default for how items are compared to their targets.
Store timestamp	Stores a timestamp along with the data any time the group executes.
Store quality code	Stores an aggregate quality for the group along with the regular data. The aggregate quality is a bit-wise AND of the qualities of the items in the group.

4.6.2.2 Trigger and Handshake Settings

The trigger settings determine when a group will actually execute. They are examined each time the group evaluates (according to the *update rate* of the group). If they pass, the group will run and perform its action against the database.

The trigger settings are the same for all group types. They are found on the second tab (labeled "Trigger"), in the right side of the Transaction Group workspace.

© 2011 Inductive Automation

Only execute when value have changed (asynchronous trigger)
These settings are evaluated first. If set, the group will examine whether the values in the specified tags have changed, and if not, will exit evaluation.

It is possible to monitor all *Run-Always* tags in the group, or only specific ones.

Execute this group on a trigger
Enables trigger on a specific item in the group. The trigger item can be any *Run-Always* item, such as an OPC item, SQLTag reference, or an Expression item set to "Run-Always" mode.

In addition to the numeric settings that define the trigger, there are several other options:

Only execute once while trigger is active - The group will only execute once when the trigger goes into an active state, and will not execute again until the trigger goes inactive first. If unselected, the group will execute each time the trigger conditions evaluate to true.

Reset trigger after execution - If using the ">0" or "=0" trigger modes, the trigger can be set to write an opposite value after the group has executed successfully. This is useful for relaying the execution back to the PLC.

Prevent trigger caused by group start - If selected, the group will not execute if the trigger is active on the first evaluation of the group. In the course of designing a group, it is common to stop and start it many times, and sometimes it is not desirable to have the group execute as a result of this. Selecting this option will prevent these executions, as well as executions caused by system restarts.

Handshake Settings
Group handshakes are also defined on the trigger tab. It is possible to specify both a success and failure handshake. The success handshake will write the specified value to the given item when the group has finished all other triggered execution without error.

The failure handshake, on the other hand, will be written when the group execution is cut short due to an error, such as an error writing to the database or an item.

4.6.2.3 Advanced Settings

Transaction groups offer several advanced settings that affect how execution occurs. These settings can be found under the **Options** tab for a group.

OPC Data Mode
This setting modifies how the group receives data from OPC.

Subscribe - Data points are registered with the OPC server, and data is received by the group "on-change". This is the default setting and generally offers the best performance, as it reduces unnecessary data flow and allows the OPC server to optimize reads. However, it's important to note that data is received by the group *asynchronously*, meaning that it can arrive at any time. When the group executes, it "snapshots" the last values received and uses those during evaluation. If some values arrive after execution begins, they will not be used until the following execution cycle.

Read - Each time the group executes it will first read the values of OPC items from the server. This operation takes more time and involves more overhead than subscribed evaluation, but ensures that all values are updated together with the latest values. It is therefore commonly used with batching situations, where all of the data depends on each other and must be updated together. It's worth

© 2011 Inductive Automation

noting that when using an OPC item as the trigger, the item will be subscribed, and the rest of the values read when the trigger condition occurs. Note: This option was previously referred to as "polled reads" in earlier versions of the software.

Bypass Store and Forward System
Only applicable to groups that insert rows into the database. Causes groups to target the database directly instead of going through the store-and-forward system. If the connection becomes unavailable, the group will report errors instead of logging data to the cache.

Override OPC Subscription Rate
Specifies the rate at which OPC items in the group will be subscribed. These items are normally subscribed at the rate of the group, but by modifying this setting it is possible to request updates at a faster or slower rate.

4.6.2.4 Items Types

4.6.2.4.1 Overview

Items are the core elements of a group. They are executed, and the values are then used by the group for logic purposes, by other items, and to write to the database. They can be written to from the database or from other items.

Type of Item	Description
OPC Item	Directly subscribed to an OPC server at the rate of the group. Executed by the group, so alerts are evaluated when the group is executed. These items are executed even when the trigger isn't active.
Run-Always Expression Item	Much like an expression SQLTag, can be either a static value, an expression, or a database query. Run-Always expression items are evaluated at each group interval, before the trigger state is evaluated.
Triggered Expression Item	Same as Run-Always expression items, except that they are only executed after the trigger has been evaluated and is active.
SQLTag Reference	A reference to a SQLTag. Allows a SQLTag to be used in a group like any other item type, except that the tag is evaluated by its scan class instead of by the group. See *SQLTags vs. OPC Items* below for more information.

Execution Order

Items generally aren't executed in a reliable order, with the exception of Expression items. Expression items can be ordered using the up and down arrows located to the right of the list where the items are displayed. This can be crucial for performing complex operations that require a specific sequence.

SQLTags vs. OPC items in Groups

It is easy to confuse the definition and purpose of SQLTags and OPC items in transaction groups, though they have distinct benefits.

SQLTags may be referenced inside of groups, however it is critical to remember that they are executed by the Ignition gateway, according to their scan classes, and independently of the group. Adding a SQLTag into a group is like creating a shortcut to that tag. However, once in the group, the item can be used like any other item. That is, it may be mapped bi-directionally to the database, used as a trigger, be the target of another item, etc. It is even possible to create an hour meter out of the item. Core properties of the tag such as alerting and scaling, however, are defined in the actual SQLTag, not in the group.

© 2011 Inductive Automation

OPC Items in groups (as well as expression items in groups), however, are completely executed by the group. They do not exist outside of the group in which they are defined. They are subscribed and evaluated according to the rate of the group.

Generally speaking, it is most common to create OPC items in groups, even if a particular point might already exist in SQLTags. This leads to more understandable group execution, as all evaluation occurs in the group according to the timer and trigger settings. SQLTag references are useful when it is necessary to have a single value in multiple groups, for example, as a trigger in order to coordinate execution.

4.6.2.4.2 OPC Item

OPC Items are the backbone of a group. They get their values from PLCs and the values are then used by other items the group and/or to write to the database. They are directly subscribed to an OPC server at the rate of the group and are executed by the group so their alerts are evaluated when the group is executed. These items are executed even when the trigger isn't active.

OPC Item Properties

General:

General Properties

Name - The name of the OPC item in the group. There cannot be duplicate names within a group.

Datatype - The datatype used to read values from the PLC.

OPC Properties

Clicking on the **Browse OPC...** button in this section will allow you to select the tag you want and Ignition will fill out the following fields for you.

OPC Server - The Selected OPC Server. This is a dropdown list showing all the OPC Servers added in the Ignition Gateway.

OPC Item Path - The OPC Item's address Including Device name or Channel if required.

Value Mode

Property - Which property of the OPC item you want to use.

1) Value - Item value
2) Quality - Quality code from OPC Server (192 = GOOD_DATA)
3) Timestamp - The last time the item value changed
4) Name - The SQLBridge Item Name property of this Item.

Mode - Options for displaying values based on the Item value.

1) Direct Value - Item value
2) Hour Meter - Record the amount of time the Item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The datatype should be set to integer or float when using an Hour Meter regardless of the OPC Item type.
   On Zero - Use a zero value to accumulate time instead of a non-zero value
   Retentive - Retain the Hour Meter value when it is not accumulating.
   Units - The time units to display.
3) Event Meter - Record the number or times the Item value is non-zero. The datatype should be set to integer when using an Event Meter regardless of the OPC Item type.
   On Zero - Use a zero value to accumulate events instead of a non-zero value

Write Target

Mode - Changes the items directional read/write option.

© 2011 Inductive Automation

1) Use group's mode - Inherit the Update Mode from the Item's Group.
2) OPC to DB - Only read from the OPC server and write to the database.
3) DB to OPC - Only read from the database and write to the OPC Server.
4) Bi-directional OPC wins - Read and Write to both the database and OPC Server. On group start, write OPC Server values to the database.
5) Bi-directional DB wins - Read and Write to both the database and OPC Server. On group start, write database values to the database.

Target Type - This is the selection for what the Item will write to when the group executes.

1) None, read-only item - Do not write this value to the database.
2) Database field - Write the Item value to the specified column in the database table. This list will populate with all the column names from the Group's target table after the first time the group is run.

Target Name - The name of the column in the database that this Item will write to when the group executes. The Target Name list will populate with all the column names from the Group's target table if the Target Type is Database field.

Alerting:

Alerting settings for the OPC items. See SQLTags Alerting for a full explanation.

4.6.2.4.3 Expression Item

Expression Items are used for executing comparisons, simple math and querying additional database tables. They get their values from an expression made up of static values or other items, or from SQL Queries. They can have alerts and can be executed when the trigger is active or every time the group executes.

Expression Item Properties

General:

General Properties

Name - The name of the OPC item in the group. There cannot be duplicate names within a group.

Value - The static value of this Expression item. This will be overwritten by an Expression/SQL binding.

Datatype - The datatype values are stored as.

Value Mode

Property - Which property of the OPC item you want to use.

1) Value - Item value
2) Quality - Quality code of the expression/SQL Query (192 = GOOD_DATA)
3) Timestamp - The last time the item value changed
4) Name - The SQLBridge Item Name property of this Item.

Mode - Options for displaying values based on the Item value.

1) Direct Value - Item value
2) Hour Meter - Record the amount of time the Item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The datatype should be set to integer or float when using an Hour Meter regardless of the OPC Item type.

© 2011 Inductive Automation

On Zero - Use a zero value to accumulate time instead of a non-zero value
Retentive - Retain the Hour Meter value when it is not accumulating.
Units - The time units to display.

3) Event Meter - Record the number or times the Item value is non-zero. The datatype should be set to integer when using an Event Meter regardless of the OPC Item type.
On Zero - Use a zero value to accumulate events instead of a non-zero value

Evaluation Mode

Run-always (ignore Trigger) - When selected, this causes the group to evaluate at each group interval, before the trigger state is evaluated.

Write Target

Target Type - This is the selection for what the Item will write to when the group executes.
1) None, read-only item - Do not write this value to the database.
2) Database field - Write the Item value to the specified column in the database table.
3) Other tag - Write the Expression Item's value back to an OPC item or SQLTag Reference.

Target Name - The name of the column in the database that this Item will write to when the group executes. The Target Name list will populate with all the OPC Item and SQLTag Reference names from this Group, or the column names from the Group's target table depending on the Target Type selected.

Numeric:

Numeric properties for Expression Items. See SQLTags Numeric Properties for a full explanation.

Alerting:

Alerting settings for the OPC items. See SQLTags Alerting for a full explanation.

Expression:

Expression/SQLQuery options for Expression Items. See SQLTags Expression/SQL Properties for a full explanation.

4.6.2.4.4 SQLTag Reference

SQLTag References are used just like OPC Items, adding the convenience of using a SQLTag that has already been set up with scaling and alarm data.

SQLTag Reference Properties

General:

General

© 2011 Inductive Automation

Tag Path - The path to the tag being referenced. This value is not editable except by clicking the Insert Tag button. There cannot be duplicate names within a group.

Data Type - The datatype to write to into the database if this item is not read-only.

Value Mode

Property - Which property of the SQLTag you want to use.

1) Value - Item value
2) Quality - Quality code of the SQLTag (192 = GOOD_DATA)
3) Timestamp - The last time the item value changed
4) Name - The SQLBridge Item Name property of this Item.

Mode - Options for displaying values based on the Item value.

1) Direct Value - Item value
2) Hour Meter - Record the amount of time the Item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The datatype should be set to integer or float when using an Hour Meter regardless of the OPC Item type.
   On Zero - Use a zero value to accumulate time instead of a non-zero value
   Retentive - Retain the Hour Meter value when it is not accumulating.
   Units - The time units to display.
3) Event Meter - Record the number or times the Item value is non-zero. The datatype should be set to integer when using an Event Meter regardless of the OPC Item type.
   On Zero - Use a zero value to accumulate events instead of a non-zero value

Write Target

Mode - Changes the items directional read/write option. This is only editable when the target Type is set to Database field.

1) Use group's mode - Inherit the Update Mode from the Item's Group.
2) OPC to DB - Only read from the OPC server and write to the database.
3) DB to OPC - Only read from the database and write to the OPC Server.
4) Bi-directional OPC wins - Read and Write to both the database and OPC Server. On group start, write OPC Server values to the database.
5) Bi-directional DB wins - Read and Write to both the database and OPC Server. On group start, write database values to the database.

Target Type - This is the selection for what the Item will write to when the group executes.

1) None, read-only item - Do not write this value to the database.
2) Database field - Write the Item value to the specified column in the database table.

Target Name - The name of the column in the database that this Item will write to when the group executes. The Target Name list will populate with all the column names from the Group's target table if the Target Type is Database field.

4.6.3 Execution Cycle

All of the groups follow a similar execution cycle. The core evaluation may differ, but the general cycle is the same.

1) Timer executes, group enters execution
2) Is the group paused? Break execution.
3) Is the Gateway the cluster's master? If not, break execution. Transaction groups only execute on the master.
4) Evaluate "run-always" items: OPC items, SQLTag references, and Expression items set to ignore the trigger.
5) Is trigger set/active? If there is a trigger defined, but it is not active, break execution.

© 2011 Inductive Automation

6) Evaluate "triggered" items: Expression items not set to ignore the trigger.
7) If applicable, read values from the database
8) Execute a comparison between items and their targets
9) Execute any writes to other Tags or the Database that result from execution.
10) Report alerts
11) Acknowledge the trigger, if applicable.
12) Write handshake value, if applicable.

If an error occurs at any stage besides the last stage, execution will break and the failure handshake will be written if configured. The group will attempt execution again after the next update rate period.

4.6.4 Types Of Groups

4.6.4.1 Standard Group

The *standard group* is called such because it's a flexible, general use group that can be adapted to a variety of situations. The data model is *row based*, with items mapping to columns and the data corresponding to a specific row of a table.

General Description

The standard group contains items, which may be mapped to the database, or used internally for features such as triggering or handshakes. Items that are mapped to the database target a specific column of a single specific row, chosen according to the group settings. Items can be mapped in a one-way fashion, or bi-directionally, in which the value of the database and the item will be synchronized.

The group may also insert new rows instead of updating a specific row. In this manner, data can be inserted for historical purposes based on a timer, with an optional trigger.

Group Settings

The standard group uses a timer-based execution model shared by all groups, and the normal trigger settings.

Additionally, there are several settings specific to the group type:

Automatically create table - If the target table does not exist, or does not have all of the required columns, it will be created/modified on group startup. If not selected and the table doesn't match, an error will be generated on startup.

Store timestamp - Specifies whether or not to store a timestamp with the record, and the target column. The timestamp will be generated by the group during execution. For groups that update a row, the timestamp will only be written if any of the values in the group is also written.

Store quality code - If selected, stores an aggregate quality for the group to the specified column. The aggregate quality is the combined quality of all of the items that write to the table. For more information about quality values, see Data Quality

Delete records older than - If selected, records in the target table will be deleted after they reach the specified age. This setting is useful for preventing tables from growing in an unbounded manner, which can cause disk space and performance problems over time.

© 2011 Inductive Automation

Table action - This section details how the group interacts with the table. The group can insert a new row each execution, or update the first, last or custom record. A custom update clause is essentially the where clause of the SQL query that will be generated to read and write the group. In addition to standard SQL syntax, you can bind to items in the group in order to inject dynamic values.

Typical Uses

Standard groups can be used any time you want to work with a single row of data. This can include:

Historical logging - set the group to insert new records, and log data historically either on a timer, or as the result of a trigger. Flexible trigger settings and handshakes make it possible to create robust transactions.

Maintain status tables - Keep a row in the database updated with the current status values. Once in the database, your process data is now available for use by any application that can access a database, dramatically opening up possibilities.

Manage recipes - Store recipe settings in the database, where you have a virtually unlimited amount of memory. Then, load them into the PLC by mapping DB-to-OPC using a custom where clause with an item binding in order to dynamically select the desired recipe.

4.6.4.2 Block Group

The *block group* is so named because it writes "blocks" of data to a database table, consisting of multiple rows and columns.

General Description

A block group contains one or more block items. Each block item maps to a column in the group's table, and then defines any number of values (OPC or SQLTag items) that will be written vertically as rows under that column. The values may be defined in the block item in two modes. The first, List mode, lets a list of value-defining items to be entered. These value items may either by OPC items, SQLTag items, or static values. The second mode, Pattern mode, can be useful when OPC item paths or SQLTag paths contain an incrementing number. You may provide a pattern for the item's path, using the wildcard marker `{?}` to indicate where the number should be inserted.

Block groups are very efficient, and can be used to store massive amounts of data to the database (for example, 100 columns each with 100 rows- 10,000 data points- will often take only a few hundred milliseconds to write, depending on the database). They are also particularly useful for mirroring array values in the database, as each element will appear under a single column, and share the same data type.

Like the standard group, the block group can insert a new block, or update the first, last or a custom block. Additionally, the group can be set to only insert rows that have changed in the block.

In addition to block items, the group can have other OPC items, SQLTag references, and Expression items. These items can be used for triggers, handshakes, etc. They may also target a column to be written, and will write their single value to all rows in the block.

Group Settings

Beyond the differences in the data, namely that the block group works with multiple rows instead of just 1, this group type shares many similarities with the Standard Group.

© 2011 Inductive Automation

The unique settings are:

Store row id - Each row will be assigned a numeric id, starting at 0. If selected, this id will also be stored with the data.

Store block id - If selected, an incremental block id will be stored along with the data. This number will be 1 greater than the previous block id in the table.

Insert new block vs. Insert changed rows - If "insert new block" is selected, each row of the block will be inserted when the group executes, even if the data has not changed. By contrast, "insert changed rows" will only insert the rows that have new data. The latter mode is particularly useful for recording history for many data points on a "on change" basis, provided there is a unique id column defined. The "store row id" feature is useful for this, as well as the ability to reference the item path in an item's value property.

Update Custom block - Like standard groups, this setting allows you to target a specific section of the table, using SQL where clause syntax, with the ability to bind to dynamic item values. Unlike standard groups, however, the where clause specified should result in enough rows to cover the block. Excess rows will not be written to, but fewer rows will result in a group warning indicating that some data could not be written.

Typical Uses

Block groups are useful in a number of situation where you need to deal with a lot of data efficiently.

Mirroring/Synchronizing array values to DB - Arrays are often best stored vertically, which makes them perfect for block groups. Pattern mode makes configuration a breeze by allowing to you specify the array as a pattern, and set the bounds.

Recipe management - Like standard groups, but when set points are better stored vertically than horizontally.

Vertical history tables - Group data points by data type (int, float, string), create a copy of the item that stores item path, and then use the insert changed rows option to create your own vertically storing historical tables. Create additional copies of the block item that refer to quality and timestamp in order to get further information about the data point.

4.6.4.3 Historical Group

The historical group makes it easy to quickly log data historically to a SQL database.

General Description

The historical group inserts records of data into a SQL database, mapping items to columns. Full support for triggering, expression items, hour & event meters and more means that you can also set up complex historical transactions. Unlike the standard group, the historical group cannot update rows, only insert. It also cannot write back to items (besides trigger resets and handshakes).

Group Settings

The settings of the historical group are identical to the settings in the Standard Group, but limited to inserting rows.

Typical Uses

Basic historical logging - Recording data to a SQL database gives you incredible storage and querying capabilities, and makes your process data available to any application that has DB access.

© 2011 Inductive Automation

Shift tracking - Use an expression item to track the current shift based on time, and then trigger off of it to record summary values from the PLC. Use a handshake to tell the PLC to reset the values.

4.6.4.4 Stored Procedure Group

The *stored procedure group* lets you quickly map values bi-directionally to the parameters of a stored procedure.

General Description

The stored procedure group is similar to the other groups in terms of execution, triggering, and item configuration. The primary difference is that unlike the other group types, the target is not a database table, but instead a stored procedure.

Items in the group can be mapped to input (or inout) parameters of the procedure. They also can be bound to output parameters, in which case the value returned from the procedure will be written to the item. Items can be bound to both an input and output at the same time.

Group Settings

The stored procedure group's settings look and act the same as those of the Historical Group. The primary difference, of course, is that instead of specifying a table name and column names, you'll specify parameter names.

Parameters may be specified using either **parameter names** or **numerical index.** That is, in any location where you can specify a parameter, you can either use the name defined in the database, or a 0-indexed value specifying the parameter's place in the function call. **Important:** You cannot mix names and indices. That is, you must consistently use one or the other.

If using parameter names, the names should not include any particular identifying character (for example, "?" or "@", which are used by some databases to specify a parameter).

Typical Uses

Call stored procedures - The stored procedure group is the obvious choice when you want to bind values to a stored procedure. It can also be used to call procedures that take no parameters (though this can also be accomplished from Expression Items/SQLTags.

Replace RSSQL - The stored procedure group is very popular among users switching from RSSQL, given that application's heavy use of stored procedures.

Known Issues

When using Oracle, you must use indexed parameters.

4.7 Windows & Components

4.7.1 Introduction

Windows and Components

Windows and components are the fundamental building blocks for projects using the Ignition Vision module. A Vision project is a collection of Windows. These windows get loaded into the Vision Client,

© 2011 Inductive Automation

where any number of them may be open at once. A window itself is a hierarchy of components. Components range in complexity from the humble Button and Label, all the way to the powerful Easy Chart and Table components.

Windows and components are designed visually with a drag-and-drop interface in the Ignition Designer. Components each have a host of properties that govern how the component looks and behaves. Components are brought to life through the combination of property binding and event handlers. These concepts should be generally familiar to anyone who has used a programming or RAD tool like Visual Basic or MS Access. Property binding is the technique of binding a component's property to something else that is changing, such as a SQLTag or the results of a database query. Event handlers are a way to use scripting to react to *events* that the component fires, such as mouse or keyboard events.

The Window Workspace

When a window is selected in the Ignition Designer, the *window workspace* will become visible. Inside this workspace are all of the windows that are currently open. Each open window gets its own editing workspace, and you switch between windows with the tabs on the bottom. It is also standard to have a component palette panel and the property editor panel open.

Whenever you hit **Save** in the Designer, all open windows are committed and the whole project is saved. Note that even when working in other workspaces, for example the Transaction Group Workspace, any open windows will be committed and saved when you hit **Save**.

Whenever a project resource that is applicable in the Client scope, such as a Window or the Client Scripting configuration is changed, all running clients get an update notification.

Preview Mode

The window workspace operates in two distinct modes: design mode and preview mode. You may switch between these modes with the play/stop buttons in the toolbar or the `Project > Preview Mode` menu item. You may also use the `F5` key to toggle between the two modes.

In design mode, your mouse is used to manipulate components in a window. You can select, drag, and resize them. You may alter data bindings and event script configuration. Data bindings are active in design mode, but event handlers are not.

In preview mode, you are interacting with a "live" version of the window. Property bindings and event handlers will run, just like in the Client.

Preview mode is useful for a quick check of the operation of a window, but it becomes cumbersome when trying to test a whole project. For that, we recommend having a launched Client up as well, and doing testing in the true Client. You can quickly launch a client in one of the three launch modes via the `Tools > Launch Project` menu.

4.7.2 Windows

4.7.2.1 Windows Overview

Creating Windows

Creating windows is a easy as pressing the **New Window** button in the toolbar, or by navigating to the `File > New > Window` menu. The dropdown on the new window button pops up a dialog box that helps you design the initial size of a window, but this is rarely necessary because of the support for multiple resolutions and the typical navigation strategy employed by Ignition Vision.

© 2011 Inductive Automation

Naming and Organization

Use the Project Browser panel to rename a window by right-clicking on it and choosing Rename, or by pressing F2. You can also create folders to organize your windows. A window's name must be unique among the windows in its folder. A window's name and folder path is very important - it will be how other windows reference it.

Window Notes

Through the right-click menu on a window in the Project Browser you can access the window's notes. This free-form text field is provided to let the designer document the purpose and any technical information about how the window works.

Importing and Exporting

You may import and export windows to external files by using the right-click menu in the Project Browser. Simply select the windows in the export wizard that you'd like to export, and choose a path for the resulting `*.vwin` file.

4.7.2.2 Anatomy of a Window

Name and Path

Windows are the top-level unit of design for Vision projects. A window is identified by its path, which is the name of all its parent folders plus its name, with forward slashes (`/`) in between. For example, the path to a window in the top level called `MainWindow` would simply be its name, whereas the path to a window named `UserOptions` under a folder called `OptionsWindows` would be: `OptionsWindows/UserOptions`.

Titlebar and Border

A window may display a titlebar and/or a border. The titlebar allows the user to drag the window around, and houses the window's close and maximize/restore buttons. The border of a window can be used to resize the window when it is floating or docked. Whether on not the titlebar and border are displayed depends on the values of the window's titlebar and border display policy properties, and its current state. Commonly, a window will display both a titlebar and border when it is floating, but only a titlebar when maximized. It is often desirable to remove titlebars and borders on maximized windows.

Root Container

Inside a window is always the *root container*. This is a normal container component except that it cannot be deleted or resized - its size is always set to fill the entire window. The root container is where you will place all of your components in the window.

4.7.2.3 Typical Window Types

By manipulating a window's properties, you can transform it into various configurations. Typically, you'll alter the window's *Dock Position*, *Border Display Policy*, *Titlebar Display Policy*, and *Start Maximized* properties to change windows into one of three categories.

Screens

A "screen" window is one that is set to *start maximized*, and has its *border* and *titlebar* display policies set to `When Not Maximized` or `Never`. This will make the window take up all available space (minus space used by any "docked" windows). This makes the window act much like a typical "HMI screen." You may also see these referred to as "main" windows, typically when referring to the currently visible one.

© 2011 Inductive Automation

Docked Windows

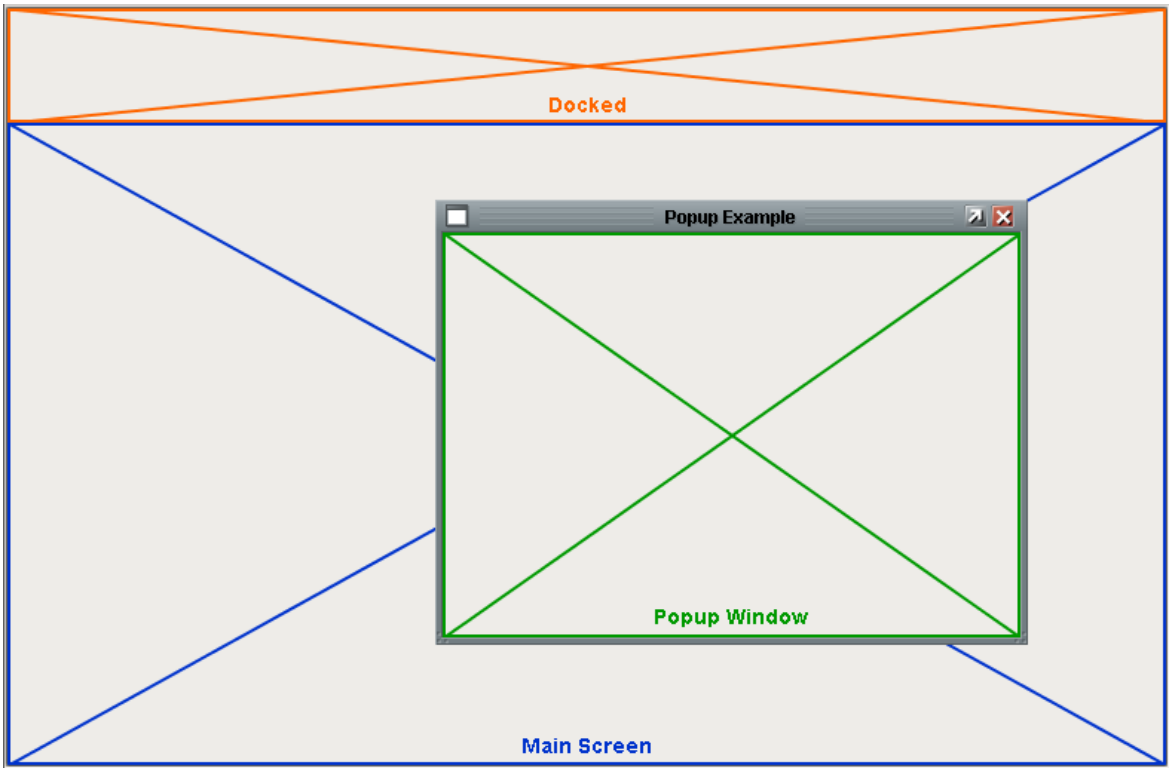
A "docked window" is one whose `Dock Position` is set to anything but `Floating`. This will make the window stick to one side of the screen, and nothing can overlap it. It will also typically have its *border* and *titlebar* display policies set to `Never`. This makes the "docked" window appear to be joined seamlessly with the current "screen" window.

These screens are usually tall and skinny or short and wide, depending on the side they're docked to. The purpose of a docked window is to make some information always available; typically navigation controls and overall status information. Using docked windows can help eliminate repetitive design elements from being copied to each screen, making maintenance easier.

Popup Windows

A "popup window" is a window whose `Dock Position` is set to `Floating` and is not maximized. Its *border* and *titlebar* display policies are usually set to `When Not Maximized` or `Always`, so that they can be manipulated by the end-user. This is how all windows start out when first created. These windows are often opened by components in the current "screen" window, and are meant to be on top of the screen. To this end, they may have their *Layer* property set to a number higher than zero so they don't get lost behind the "screen" window. To get a window to pop-up at a specific position, edit the Window's `Starting Location` property.

Popup windows are often *parameterized* so they can be re-used.



See also:
Typical Navigation Strategy
Parameterized Windows

© 2011 Inductive Automation

4.7.2.4 Window Properties

Special Properties

Windows have some special properties that you can edit while the window is closed. These properties are modified by right-clicking on the window in the Project Browser.

Name	The name of the window. Must be unique in its folder.
Open on Startup	Windows with this property set to true will be opened when the project starts up in the Vision Client.
"About" Window	At most one window per project may specify an "about" window. This will cause an "About this Application" menu item to appear in the "Help" menu in the Client, which opens the appropriate window.

Dynamic Startup Windows
Sometimes a project needs to alter its startup windows depending on who logged in, what security roles the have, or what computer the client is launched on. In these cases, simply set no startup windows, and write a Client Startup Script that uses the `system.nav` library to open the correct windows.

Standard Properties

These properties are modified in the Property Editor panel, just like a component's properties. Simply select the window either by clicking on its title bar, or clicking on the window's node in the Project Browser while it is open to select it in the Property Editor.

Appearance

Title
The title to be displayed in this window's titlebar.

Scripting name	`title`
Data type	String

Border Display Policy
Determines if window's border is shown in various window states.

Scripting name	`borderDisplayPolicy`
Data type	int
Values	0 Always 1 Never 2 When Not Maximized

Titlebar Display Policy
Determines if window's titlebar is shown in various window states.

Scripting name	`titlebarDisplayPolicy`
Data type	int
Values	0 Always 1 Never 2 When Not Maximized

Titlebar Height
The height of the window's titlebar.

Scripting name	`titlebarHeight`
Data type	int

Titlebar Font
The font of the window title in the titlebar.

Scripting name	`titlebarFont`
Data type	Font

Behavior

Dock Position
Determines the position this window is docked to, or if it is floating.

© 2011 Inductive Automation

Scripting name `dockPosition`
Data type int
Values
0 Floating
3 West
4 South
2 East
1 North

Closable

Determines whether or not to draw the close (X) button in the upper right corner.

Scripting name `closable`
Data type boolean

Maximizable

Determines whether or not to draw the maximize button in the upper right corner.

Scripting name `maximizable`
Data type boolean

Resizeable

Determines whether or not to let the user resize the window.

Scripting name `resizable`
Data type boolean

Start Maximized

When set to `true`, the window will become maximized when it is opened.

Scripting name `startMaximized`
Data type boolean

Cache Policy

By default this property is set to `Auto`, which keeps a window in a memory cache for a while after it is closed, so that if it is opened again it will be quick. The window isn't "active" while it is closed: all of its bindings and scripts are shut down.

Setting this property to `Never` causes a fresh copy of the window to be deserialized every time it is opened. This is a performance hit, but it also is a convenient way to "clear out" the values of the window from the last time it was opened, which can be helpful in data-entry screens.

Setting the property to `Always` will trade memory for higher performance, causing the window to always remain cached after the first time it is opened. This means the window will open very fast, but your Client will need lots of memory if you do this to a large amount of windows.

Scripting name `cachePolicy`
Data type int
Flags expert
Values
0 Auto
1 Never
2 Always

Layout

Location

The location that this window will open up at. Only applicable to floating windows that are not set to start maximized. Also, you must un-check the "Center Window" checkbox on the open-window navigation action in order for this location to take effect

Scripting name `startingLocation`
Data type Point

© 2011 Inductive Automation

Size	The dimensions of the window. This can be manipulated by selecting the window and dragging the resize handles along the windows right and bottom edges. Scripting name `size` Data type Dimension
Minimum Size	The minimum size that this window will allow itself to be resized to. Scripting name `minimumSize` Data type Dimension Flags expert
Maximum Size	The maximum size that this window will allow itself to be resized to. Scripting name `maximumSize` Data type Dimension Flags expert
Layer	Sets the layer that this window is in. Default layer is 0, which is the bottom layer. Windows in higher layers will always be shown on top of windows in layers beneath them. Scripting name `layer` Data type int Flags expert

4.7.2.5 Window Security

You can configure security settings that control who can and who can't open a window. While the window is open, select it by clicking on the title bar or selecting its node in the Project Browser. Then navigate to the `Component > Component Security` menu. Window security is configured the same way that Component Security is configured.

4.7.2.6 Typical Navigation Strategy

Make sure you understand the Typical Window Types topic before reading this topic.

The typical navigation strategy for a Vision project is to have a "docked" window or two (usually docked north and/or west), and then have a single "screen" style window visible at a time. *Swap* navigation is used to swap between the screens. This ensures that only one screen is open at a time. Standard *open* navigation is then used to open various "popup" windows as necessary.

This style of project is so common, that the default operation of the Tab Strip component expects it. When it is in its default automatic operation, it expects that each tab represents a "screen" window, and will automatically swap from the current screen to the desired screen. Furthermore, the `[System]/Client/User/CurrentWindow` tag is calculated based upon this strategy: its value is the name of the current maximized window.

This navigation strategy is used in the "ExampleProject" that you can download from our website.

4.7.2.7 Swapping vs Opening

There are two primary window navigation operations: *swapping* and *opening*.

Opening and Closing

Opening and closing are the basic window navigation options. Opening a window opens the window at the same size it was the Designer, unless the *Start Maximized* property is `true` or the *Dock Position* is not `Floating`. To have a floating popup window open at a specific location, make sure to set the

© 2011 Inductive Automation

Location property of the window in the Designer. If the window was recently open, it will open in its last state due to window caching. See the Cache Policy property for more information.

Swapping

In general, swapping involves closing one window, and then opening another window in its place. This operation can be performed on window in any state: docked or floating, maximized or not. The *Start Maximized* and *Dock Position* properties of the window that is being swapped in will be ignored - it will take the dock and maximized state of the window that it is replacing.

This operation is so common in the typical navigation strategy that there is even a version of swapping dedicated to it, the `swapTo` function. This function eliminates the need to specify the window to swap from - you only need to specify the window to swap to. It will take the current "screen" window - that is, the current maximized window - as the window to swap from.

See also:
system.nav.openWindow
system.nav.swapWindow
system.nav.swapTo

4.7.2.8 Open Windows and Performance

While a window is open, its query bindings are running, its tag bindings are keeping tags subscribed, and its event scripts are being executed. This means that an open window is actively using system resources, both on the Client's host machine, and on the Gateway's server machine as its queries and tag subscriptions must be handled. For these reasons, it is important that you properly implement a navigation strategy that prevents windows that are no longer being used from being held open.

The most common mistake that will cause windows to stay open unintentionally is to implement a swapping navigation system using the `swapTo` function on windows that are not maximized. When you do this, the `swapTo` function cannot calculate the window to swap *from*, thereby simply opening the window, and not closing any windows. It is easy to check the Windows menu to see what windows are currently open. If there are more windows listed there than you can currently see, there is a problem in your navigation logic that is failing to close windows properly.

4.7.2.9 Parameterized Windows

It is often useful to create a *parameterized* window that can be re-used for multiple purposes, depending on the values that were passed into it when it was opened. For example, suppose you have 10 compressors, and the tags that represent them are predictable based upon the compressor number.

```
Compressors/
   C1/
      HOA
      Amps
   C2/
      HOA
      Amps
   ...
   C10
      HOA
      Amps
```

You could make a single compressor status & control screen, and simply pass the relevant compressor number to it when you open it.

© 2011 Inductive Automation

Passing Parameters
Any dynamic property on the root container of a window can be used as a window parameter. Simply specify the names of the dynamic properties to set in the call to `openWindow` to use them as parameters. Then, use the dynamic property to create indirect property bindings that bind to the appropriate spot.

For example, let's suppose that you had a window called `CompressorPopup` that you wanted to use to control all 10 compressors. You'd put a dynamic property on your compressor control window called `compNum`. You would use `compNum` in your tag bindings for the controls on your screen using indirect tag bindings. For example, you might bind the **control** and **indicator** properties of a Multi-State Button to an indirect tag binding like:

```
Compressors/C{1}/HOA
```

where the `{1}` paremeter is bound to the property path:

```
Root Container.compNum
```

You could use a similar indirect binding to display the amperage in an analog Meter component.

Now, when opening the window, you could use a script like this to open it to control compressor #6. Of course, you probably wouldn't write this script by hand, you'd use the navigation script builder. But it is useful to know what the script would look like.

```
system.nav.openWindow("CompressorPopup", {"compNum":6})
```

Opening Many Copies
By default, opening a window will only ever open one copy of any given window. If the window is already open, it simply brings it to the front. Normally this is the desired behavior. For example, if you opened the compressor popup window for compressor #6, and then opened it for compressor #4, the window that had been controlling #6 will switch to controlling #4.

Sometimes you may want to open a separate popup, one for #6, and one for #4, both at the same time. If this is the case, use the system.nav.openWindowInstance function call to open your window.

4.7.3 Components

4.7.3.1 Introduction

Components are what fill up your windows with useful content. Anyone familiar with computers should already understand the basic concept of a component - they are the widgets that you deal with every day - buttons, text areas, dropdowns, charts, etc. The Vision module comes with a host of useful components out of the box, many of which are specialized for industrial controls use. Other modules, like the Reporting module, add more components for specialty purposes.

Configuring components will likely be the bulk of a designer's work when designing a Vision project. The basic workflow is to take a component from the palette and drop it into a container on a window. From there, you can use the mouse to drag and resize the component into the correct position. While the component is selected, you can use the Property Editor panel to alter the component's properties, which changes the component's appearance and behavior.

To make the component do something useful, like display dynamic information or control a device register, you configure property bindings for the component. To make the component react to user interaction, you configure event handlers for it.

© 2011 Inductive Automation

4.7.3.2 Creating Components

4.7.3.2.1 Component Palette

Choose your palette

There are two styles of component palette in Ignition Vision: the tabbed palette and the collapsible palette. These palettes work in the same way, but the tabbed palette is meant to dock to the north or south edge of the workspace, and one collapsible palette is meant to dock to the east or west edge. By default, the tabbed palette is visible in the window workspace. To switch palettes, navigate to the `View > Panels` menu, and select either the `Tabbed Palette` or the `Collapsible Palette`.

Create a component

There are two primary mechanisms for creating components:

1. Select the component in the palette, and then use the mouse to draw a rectangle in a container.
   While a component is selected in a palette, the mouse curser will be a crosshair (┼) when hovering over a container that the component can be dropped in. Draw a rectangle in the container to specify where the component should be placed and what size it should be.
2. Drag a component's icon from a palette onto a container. The component will be placed where you dropped it at its default size. It can then be resized.

4.7.3.2.2 Custom Palettes

Custom palettes are like expanded copy/paste clipboards. They allow you to put customized components or groups of components into a palette for quick access.

To create a custom palette, right click on a tab in the tabbed palette or a header in the collapsible palette, and choose *New Custom Palette*. Your custom palette will appear as the last palette. Your custom palette has one special button in it, the capture button ( ). To add components to your palette, select them and press the capture button. This effectively does a copy, and stores the captured components as a new item in the clipboard. You can then use that item much like a normal component, and add multiple copies of it to your windows.

Note that these are simple copies, and are not linked back to the custom palette. Re-capturing that palette item will not update all uses of that item across your windows.

4.7.3.2.3 SQLTags Drag-n-Drop

Components can also be created by simply dragging a SQLTag onto a container. Depending on the datatype of the tag, you will get a popup menu prompting you to select an appropriate type of component to display or control that tag.

For example, suppose you have an Int4 type tag, If you drag the tag from the SQLTags Browser panel onto a component, you will be prompted either to display or control the tag with a variety of labels, level indicators, numeric entry fields, and control buttons.

This technique is great for beginners and for rapid application design. By dropping a SQLTag into a container and choosing a component type, a few steps are happening:

The component that you chose is created at the position you dropped it.
A variety of property bindings are created automatically. The bindings depend on what kind of tag was dropped and what kind of component was created. For example, lets suppose you have a `Float8` point that represents a setpoint, and you want to set it. Drop the tag onto a container and choose to control it with a *Numeric Text Field*. The following bindings will be set up automatically

© 2011 Inductive Automation

- The text field's `doubleValue` property gets a *bidirectional* tag binding to the tag's `Value` property.
- The text field's `minimum` and `maximum` properties get tag bindings to the tag's `EngLow` and `EngHigh` properties, respectively.
- The text field's `decimalFormat` property gets a tag binding to the tag's `FormatString` property.
- The text field's `toolTipText` property gets a tag binding to the tag's `Tooltip` property

It is important to realize that multiple property bindings are created when creating components this way. These bindings not only use the tag's value, but much of the tag's *metadata* as well. Using the tags metadata in this way can greatly improve a project's maintainability. For example, if you decide that the setpoint needs 3 decimal places of precision, you can simply alter the tag's `FormatString` to be `#, ##0.000`, and anywhere you used that tag will start displaying the correct precision because of the metadata bindings.

See also:
Property Binding Overview
SQLTag Metadata Properties

4.7.3.3 Selecting Components

There are a number of different ways to select components within a window, each of which have their own advantages.

Mouse Selection

Using the mouse is the most common way to select components. Simply click on a component to select it. If the component you want to select is obscured by other components, hold down `alt` and keep clicking, the selection will step down through the z-order.

You can also select components using *window-selection*. Click-and-drag in a container to draw a selection rectangle. If you drag the window left-to-right, it will select all components that are contained within the rectangle. If you drag the window right-to-left, it uses *window-crossing selection*. This will select all components that are contained within the rectangle or intersect the edge of the rectangle. Lastly, you can start dragging a window selection and then hold down the `alt` key to use *touch-selection*. This will draw a line as you drag, and any components that the line touches will become selected. As you're using these techniques, components that are about to become selected will be given a yellow highlight border.

Tree Selection

By selecting nodes in the project browser tree you can manipulate the current selection. This is a handy way to select the current window itself, which is hard to click on since it is behind the root container. (you can click to it though, using alt-click to step down through the z-order). It is also the only way to select components that are invisible.

4.7.3.4 Manipulating Components

Manipulating components can be done with both the mouse and the keyboard. To manipulate a component or group of components, you'll first need to select them.

Resizing

Once the components you want to alter are selected, they'll gets 8 *resize-handles* displayed around the edge of the selection. These handles look like double-sided arrows around the perimeter. Use the mouse

© 2011 Inductive Automation

to drag them to change the size of the components in the selection. To maintain the selection's aspect ratio, hold down `ctrl` as you resize. To resize around the center of the current selection, hold down `shift`.

You can also resize the current selection using the keyboard. To nudge the right or bottom edge of the selection in or out, use `shift` combined with the arrow keys. To nudge the top or left edge of the selection, use `ctrl-shift` combined with arrow keys. These nudge actions will resize one pixel at a time. To resize faster, add the `alt` key.

Moving

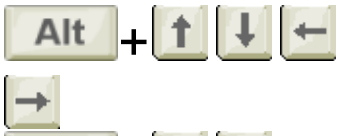
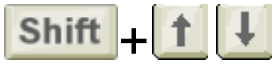
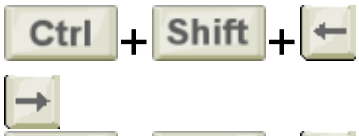
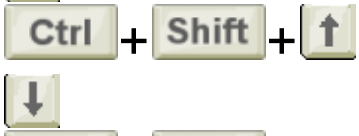
To move the component, simply drag it anywhere within the component's bounds. You can also move whatever is currently selected by holding down `alt` while dragging, regardless of whether or not the mouse is over the current selection. This is important because it is the primary way to move a Container component. (Normally, dragging in a container draws a selection rectangle inside that container).

While a component is selected, you may also use the keyboard's arrow keys to move a component around. The arrow keys will move the selection one pixel at a time. Just like resizing with the arrow keys, to move faster, add the `alt` key.

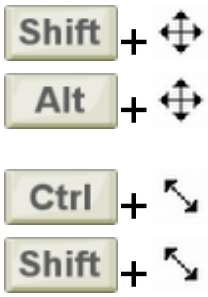
Components can be easily duplicated by dragging them as if you were going to move them and holding down the `ctrl` key. This will drop a copy of the component at the desired drop location. It is often useful to also hold down `shift` as you do this to ensure exact alignment. You may also use the `ctrl-D` shortcut to quickly duplicate a component in place.

4.7.3.5 Keyboard Shortcuts

Component Manipulation Shortcuts

	Nudge. Move selected component(s) the direction of the arrow key by the default nudge distance.
	**Alt-Nudge.** Just like nudge, but uses the "alt-nudge" distance. Set the nudge distances in Designer Window Editing Properties
	Resize Right. Moves the right edge of the component left or right. Add **`Alt`** to use the alt-nudge distance.
	**Resize Bottom.**
	**Resize Left.**
	**Resize Top**
	Move Forward / Move Backward. Move selected component(s) in the Z-order
	Move to Front / Move to Back.
	Copy-Move. Holding `ctrl` while doing a mouse move copies the current selection.

© 2011 Inductive Automation



Orthogonal-Move. Holding `shift` while doing a mouse move restricts to only moving straight up, down, left or right.
Selection-Move. Holding `alt` while dragging will drag the current selection, even if the mouse is not currently hovering over the selection or the selection isn't visible.
Proportional Resize. Resizes the selection while maintaining its aspect ratio.
On-Center Resize. Resizes the selection using the center as the anchor point.

4.7.3.6 Properties

Each component has a unique set of *properties*. A property is simply a named variable that affects something about the component's behavior or appearance. Each property has a distinct type. Hover your mouse over the property in the Property Editor panel to see its data type and scripting name.

4.7.3.7 The Property Editor

The property editor is a dockable panel that appears in the window workspace, usually under the SQLTags Browser panel. It displays the properties of the selected component. If more than one component is selected, then it will show all properties that the current selection set have in common.

Filters

It is common for components to have many properties, so the property editor by default only shows the *basic* properties. These are the properties that you'll most commonly want to set or bind for a given component. There is also the *standard* properties. This is a larger set of properties that includes the basic properties and many other useful properties. Some properties are *expert* properties. These are properties that are either uncommon to set or whose purpose might require an in-depth understanding of the inner-workings of the component. You can change the filter using the filter button ( ) in the property editor's toolbar.

Status Indication

The name of a property in the property editor conveys important information about that property:

A blue name indicates that the property is a dynamic property.

A **bold name** with a link icon indicates that the property is bound using a property binding.

A **bold name** with a color palette icon indicates that the property is being affected by the component's styles settings.

A **red bold name** with a warning icon indicates that the property is double-bound. This means that two things, a property binding and the styles settings are both trying to drive the property value. This is almost assuredly a mistake.

The Binding Button

To the right of most properties is the binding button ( ). Use this button to modify the property binding that is driving that property. You can only use this button when the window workspace is not in preview mode. Some properties cannot be bound because their datatype is not supported by the binding system. You can still use scripting to affect these properties.

4.7.3.8 Data Types

There is a wide variety of datatypes across all of the Vision Module's components. Here are the most common types that you'll find.

© 2011 Inductive Automation

Numeric Types

`Boolean`	A `true`/`false` value. Modeled as `0`/`1` in Python. Technically, `0` is `false` and anything else is `true`.
`Short`	A 16-bit signed integer. Can hold values between -2^{15} and $2^{15}-1$. Thats -32,768 to 32,767, inclusive
`Integer / int`	A 32-bit signed integer. Can hold values between -2^{31} and $2^{31}-1$. Thats -2,147,483,648 to 2,147,483,647 inclusive
`Long`	A 64-bit signed integer. Can hold values between -2^{63} and $2^{63}-1$. Thats -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 inclusive
`Float`	A 32-bit signed floating point number in IEEE 754 format.
`Double`	A 64-bit signed floating point number in IEEE 754 format.

Non-Numeric Types

`String`	A string of characters. Uses UTF-16 format internally to represent the characters.
`Color`	A color, in the RGBA color space. Colors can easily be made dynamic or animated using Property Bindings or Styles
`Date`	Represents a point it time with millisecond precision. Internally stored as the number of milliseconds that have passed since the "*epoch*", Jan 1st 1970, 00:00:00 UTC.
`Dataset`	A complex datastructure that closely mimics the structure of a database table. A Dataset is a two-dimensional matrix (a.k.a. a table) of data organized in columns and rows. Each column has a name and a datatype.
`Font`	A typeface. Each typeface has a name, size, and style.
`Border`	A component border is a visual decoration around the component's edges. You can make a border dynamic by using Styles or the toBorder expression.

Whats the difference: `Integer` *vs* `int`**?** The difference is that an `Integer` property will accept the special `null` (or `None` in Python-speak) value, while an `int` property will not. This distinction holds true for all of the numeric types: the type name that starts with a capital letter accepts null, while the all-lowercase version does not.

Expert Tip: Most of these datatypes are actually defined by Java. For example, the Date datatype is really an instance of a `java.util.Date`. This means that you can use the `java.util.Calendar` class to manipulate them, and the `java.text.SimpleDateFormat` class to format and parse them. Learn more about these classes in the Java 2 Platform online documentation at http://java.sun.com/j2se/1.5.0/docs/api/index.html

See also:
Working with Different Datatypes

4.7.3.9 Component Customizers

In addition to their properties, many components can be further customized using a *Customizer*. Many components will have more than one customizer. You can open the customizer for any component by right-clicking on it and choosing the `Customizers` menu, or by using the customizer split-button ( ) in the Vision main toolbar.

Customizers are used to configure components in ways that are too complex or cumbersome for basic properties. Some customizers are used repeatedly for many different components, for example, the Dynamic Properties customizer and the Styles customizer. Other customizers are unique for their

© 2011 Inductive Automation

component, for example the Easy Chart cutsomizer or the Report Designer customizer.

☞ *Expert Tip:* Often, a customizer is really just a user-friendly user interface to one or more expert properties. For example, all the Easy Chart customizer really does is modify the contents of the *pens, tagPens, calcPens, axes,* and *subplots* Dataset properties. Knowing this is very powerful, because this means you can also use Property Bindings and scripting to modify the values of these expert properties *at runtime*, giving you the ability to dynamically perform complex manipulations of components.

4.7.3.10 Dynamic Properties

Most Vision components support *dynamic properties.* This means that in addition to the normal properties of the component, you can add your own properties. You can think of these properties like your own variables in the window.

How to use Dynamic Properties

Dynamic properties are extremely useful and are integral to creating maintainable projects using Ignition Vision. They let you turn a "plain" component into one customized for your particular use.

To add a dynamic property, open up the *Dynamic Property Customizer* on a component that supports it. This is a simple customizer that lets you edit a table of the dynamic properties for the component. When you hit OK, you'll notice the dynamic properties you added displayed in the Property Editor in blue. You can use these properties like any others - with data binding, scripting, styles, etc.

Example
Let's look at an example that will use the dynamic properties and the Styles feature together. Take the lowly Label component. It seems pretty plain at first - it just displays a String. Of course, you can use its foreground color, background color, and border to make it look interesting. Put an integer dynamic property on it called `state`, and bind that property to a discrete state tag coming out of a PLC. Now use the `state` to drive its Styles configuration to make the component look different and display different things based on the state being 0, 1, or 2 (maybe for a Hand/Off/Auto indicator) Now, we could have used the Multi-State Indicator from the get-go, but understanding this example will let you create your own types of components by combining the existing components in creative ways.

Root Container Dynamic Properties

Properties on the window's Root Container are special in that they double as a window's *parameters*. See Parameterized Windows for more.

4.7.3.11 Component Styles

Many components support the Styles feature. This is a feature that allows you to define a set of visual styles that will change based upon a single *driving* property. Typically, you'll have a property (often a dynamic property) on your component that you want to use as a driving property, usually a discrete state, and you have multiple visual properties, like the font, border, foreground color, visibility, etc that you want to change based upon that one driving property.

Styles lets you define these relationships all at once, and with a preview to boot! Configuring styles goes like this:
1. Pick a *driving property.* This is the property whose value will determine the current style.
2. Pick one or more *styled properties*. These are the properties that will change as the style changes.
3. Add the values of the *driving property* that define the styles (e.g. 0=off, 1=hand, 2=auto)
4. Customize the values of the *styled properties* for each style.

Example
Lets say that you have a Level Indicator component that is displaying the level in a tank. Lets say that

© 2011 Inductive Automation

you want to have its appearance change based upon the alarm state of the tank's temperature. You can add an integer dynamic property to the level indicator that you'll bind to the tank temperature tag's `AlertCurrentSeverity` property.

Now go into the Style customizer. Choose your severity property as the driving property, and the Border and Filled Color properties as the styled properties. Add states for -1 (not in alarm), 2 (Medium alarm) and 4 (High alarm). Leave the -1 state alone. Use a red border for state 2 and an orange fill color. For state 4, you can animate it to get a flashing effect. Add two animation frames and set their delay to 500ms each. Configure the frames differently from each other so that you can get a flashing effect.

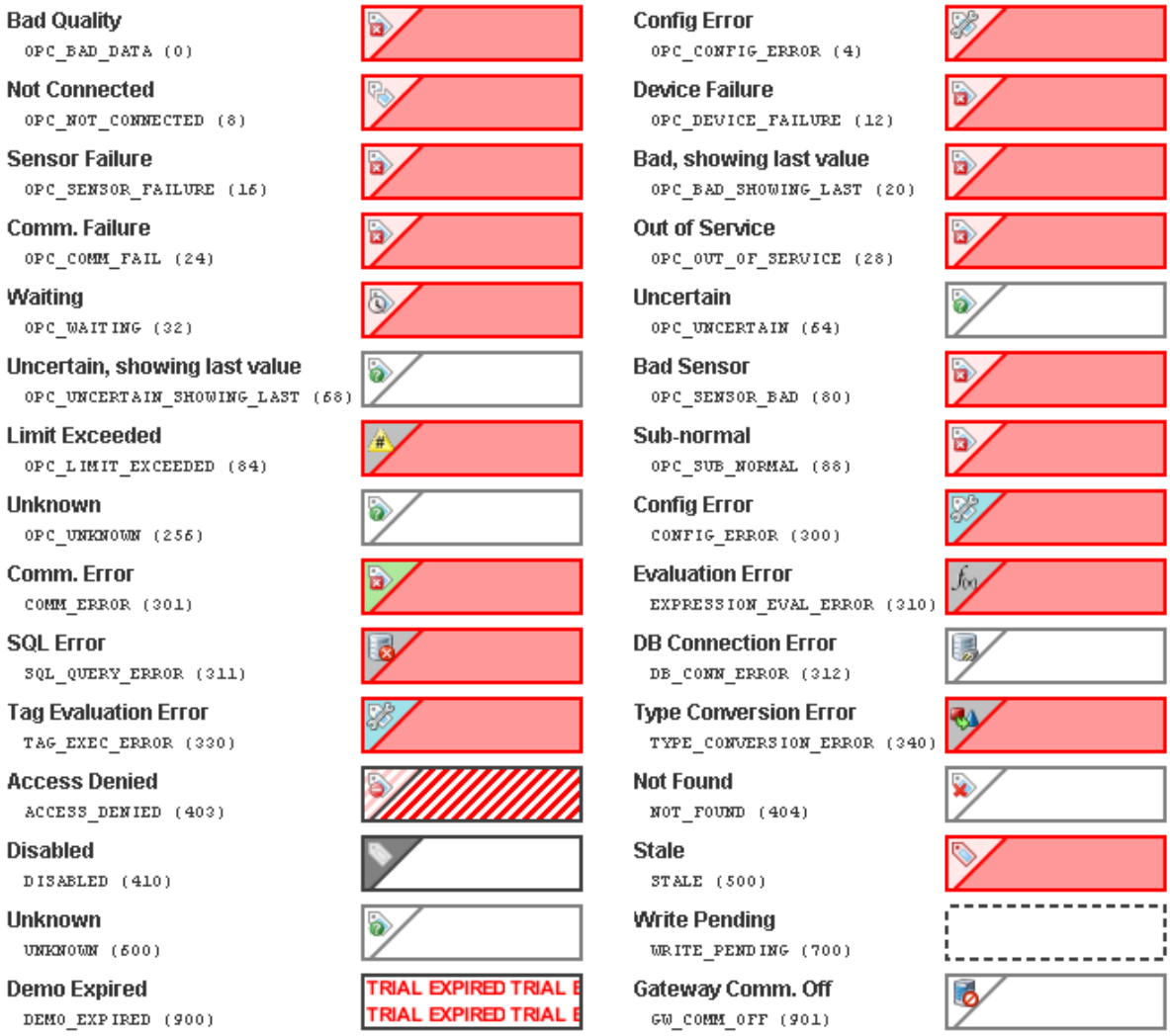
Hit OK - thats it! Notice that the styled properties that you chose are now bold and have the styles indicator ( ) next to them. This is to help remind you that those properties are being driven, so if you change their values directly, you changes will be overwritten.

4.7.3.12 Quality Overlays

Sometimes things don't go quite as expected. Connections get broken, switches die, machines crash. Aside from taking reasonable steps to prevent these occurrences, it is especially important in HMIs to be able to gauge the health and accuracy of what is displayed at a glance. In a highly distributed system like Ignition, it is especially important, as the client may be located at quite a distance (maybe across the world) from the physical process it is monitoring and controlling.

For these reasons, components will get visual overlays for various reasons to indicate that the data they are displaying is not good. Each data binding that drives a component is evaluated for quality. If any of these qualities becomes poor, the component will show an overlay. The different overlays can mean different things, denoting their underlying cause. These follow the Quality properties of SQLTags.

© 2011 Inductive Automation



4.7.3.13 Touchscreen Support

It is very common to deploy Ignition Vision projects on touchscreen computers. Often, these are industrial panel-pcs acting as HMI or OIT terminals. Normally touchscreens simply act like a mouse input device. However, these touchscreens usually don't have a keyboard attached. For this reason, all of the input components in Vision are touchscreen-enabled.

Under normal circumstances, you don't have to do anything special other than enable touchscreen-mode on your project. This will allow the operator to enable touchscreen mode when they log in. You can also enable touchscreen mode via scripting.

Touchscreen-enabled input components all have an expert level property called `Touchscreen Mode`. Normally, this is set to *Single-Click*, meaning that the touchscreen keyboard or numeric keypad (depending on the type of input component) will appear on a single click in that component. You can also change this to *Double-Click*, which should be self-explanatory, or *None*. None means that automatic touchscreen support is disabled for this component. You may want to set this to handle touchscreen logic via scripting.

To handle touchscreen logic via scripting, the general pattern is to respond to a mouse event, popup up

© 2011 Inductive Automation

a keyboard, and then set the component's value to whatever was entered in the keyboard. For example, for a text field, you would write a script like this:

```
if system.gui.isTouchscreenModeEnabled():
   currentText = event.source.text
   newText = system.gui.showTouchscreenKeyboard(currentText)
```

See also:
Client General Properties
system.gui.setTouchscreenModeEnabled

4.7.3.14 Component Layout

Layout is the concept that a component's size and position relative to its parent container's size and position can be dynamic. This allows the creation of windows that resize gracefully. This is a very important concept because of the web-launched deployment of Vision clients - they often end up being launched on many different monitors with many different resolutions.

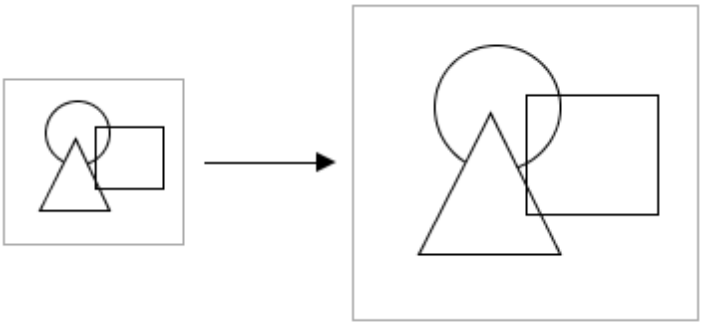
This is also important for components that have user-adjustable windows like popup windows. Imagine a popup window that is mostly displaying a large table or chart. If you're running on a large monitor, you may want to make the window bigger to see the table or chart easier. Of course, this is only useful if the table or chart actually gets larger with the window.

Changing a component's layout is as simple as right-clicking on the component and opening the Layout dialog box. You can also alter the default layout mode that gets assigned to new components. See Designer Window Editing Properties.

Relative vs Anchored Layout

There are two layout modes, and they are set on a per-component basis. Both affect the component's size and position relative to its parent container. The root container's size is dictated by the window size.

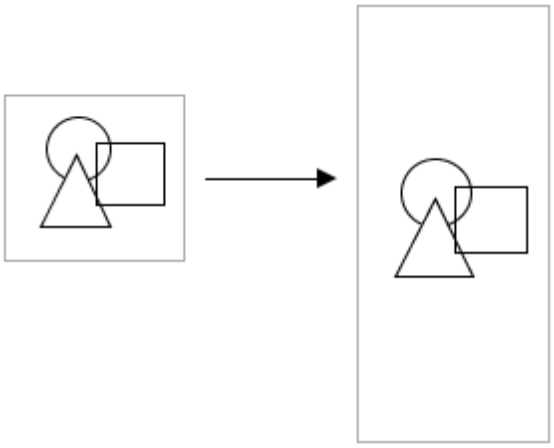
Relative Layout
Relative layout is the default mode. This is a simple but effective layout mode that simply keeps a component's size and position relative to its parent container constant, even as the parent container grows or shrinks. More precisely, it remembers the component's position and size as a percentage of its parent's bounds at the last time the window was saved. Relative layout also has the option of scaling a component's font appropriately.



Example of Relative Layout

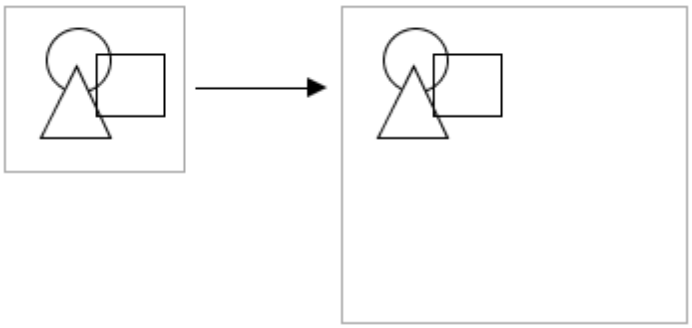
Note that relative layout mode respects aspect ratio. So if the parent component is distorted, the contents will not be. The extra space is distributed evenly on both sides of the contents.

© 2011 Inductive Automation



Relative layout preserves aspect ratio

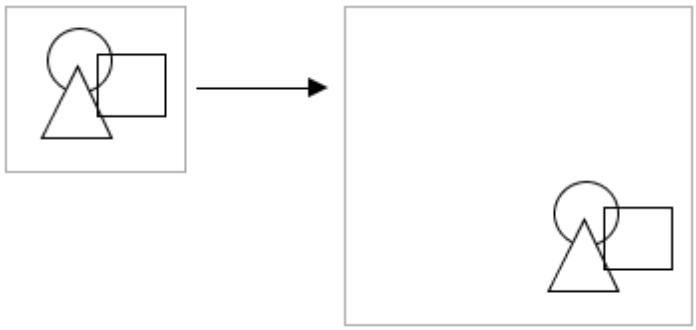
Anchored Layout
Anchored layout lets you specify various "anchors" for the component. The anchors dictate how far each of the 4 edges of the component stay from their corresponding edges in the parent container. For example, if you anchor top and left, then your component will stay a constant distance from top and left edges of its parent. Since you didn't specify an anchor for the right or bottom sides, they won't be affected by the layout.



Components anchored top and left

If you anchor bottom and right instead, the components will again stay the same size (since you didn't specify an anchor for their other edges, but they will stay a constant distance from their parent's right and bottom edges.

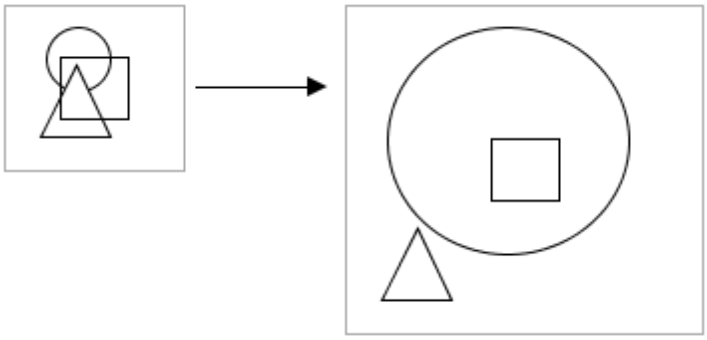
© 2011 Inductive Automation



Components anchored bottom and right

Of course, you can mix and match the various modes. There are also special centering anchors. The following image shows the following:

The square uses a horizontal and vertical centering anchor. It is centered, and stays the same size.
The triangle is anchored bottom and west.
The circle is anchored top, left, bottom, and west. This means that its edges are all anchored and stay a fixed distance to each of its parent's edges, so it grows.



Components with various anchoring modes

Client Minimum Size

Clients can define a minimum size, because even with dynamic layout, you usually don't want the client to get too small. This is because it would become unusable and unreadable. This is what the Minimum Size property is for. By default it is set to 800x600, but you can adjust it. See Client User Interface Properties.

4.7.4 Property Binding

4.7.4.1 Overview

Property Binding is perhaps the most important concept to understand when designing a project using the Vision module. It is primarily through property binding that you bring windows to life, and have them do useful things.

When you initially place a component on a screen, it doesn't really do anything. Changing its properties in the designer will make it look or act different, but it has no connection to the real world. This is what property binding adds. Property binding, as its name suggests, lets you *bind* a property to something

© 2011 Inductive Automation

else. That *something else* might be:

an OPC Tag
the results of a SQL query executed against a remote database
some other component's property
an expression involving any of these things
the results of a Python script
etc...

For example, bind the `value` property of an LED Display to an OPC SQLTag, and voilà - the `value` property will always be the value of that tag - creating a dynamic display. Bindings can also work the other way, using a *bidirectional* binding. Bind the value of a numeric text box to a tag, and that tag will be written to when someone edits the value in the text box.

The power of property bindings comes from the variety of different binding types that exist, and the fact that you can bind nearly *any* property of a component to anything else. Want it's foreground to turn red when an alarm is above a certain severity? Bind its LED Lit (`glyphForeground`) color to a tag's `AlertCurrentSeverity` property. Want it to only appear if a supervisor is on shift? Bind its `visible` property to the result of a SQL query that joins a personnel table with a shift table. The possibilities are, quite literally, endless.

How Bindings Work: Event-based vs Polling

While there are quite a few different binding types, they all boil down into two broad categories. Some complex bindings can span both categories.

Event-based bindings are evaluated when the object they are bound to changes. For example, when you bind a property to a SQLTag, that binding listens to the SQLTag, and every time the tag changes, it assigns the tag's new value into the property that it is on. If you bind the `value` of a Cylindrical Tank to the `value` of a Slider, then every time the slider changes, it fires a `propertyChangeEvent`. The binding is listening for this event, and when it is fired, updates the tank's `value`. The following bindings are event-based:

Tag bindings
Property bindings

Polling bindings are evaluated when a window first opens, on a timer, or when they change. For example, if you bind the `data` property of a Table to the results of a SQL query, that query will run on a timer, updating the Table every time it executes. The following bindings are based on polling:

SQL query bindings
some expression functions, like runScript() or now()

Many bindings can combine elements of a polling binding and event based binding. An expression binding may combine lots of other bindings to calculate a final result. A query binding will often itself be dynamic, altering the query based on other bindings.

For example, you might have a dropdown on a window that lets the operator choose a type of product that is produced. Then you can use a query binding like this to calculate the defect rate for the given product:

```
SELECT
   SUM(defective) / COUNT(*) AS DefectRate
FROM
   production_table
WHERE
   productCode = '{Root Container.ProductPicker.selectedValue}'
```

© 2011 Inductive Automation

The red code is a binding inside of the query binding. Every time this (event-based) binding fires, the query will run again.

Using bindings like this, you can create highly dynamic and interactive screens with no scripting whatsoever.

4.7.4.2 Polling Options

For bindings that poll, you have a few options.

Polling Off
A polling-off binding will execute once when the window is opened, and then it will only execute again if it changes. The typical example of a binding that can change is a SQL query binding where it uses the brace-notation (`{}`) to include dynamic information inside the query. When this dynamic information changes the query, it will run again.

Relative Rate
The binding will execute at a regular rate, based on a delta off of the project's base polling rate. See Client Polling Properties. This is usually a good idea so that you can speed up or slow down an entire client's polling system in one place.

Absolute Rate
Using this option, you can specify an absolute rate for the binding to execute at, instead of one that is based off the relative rate.

4.7.4.3 Bidirectional Bindings

Tag bindings and Query bindings can be set up as bidirectional bindings. This means that not only is the binding assigning the tag value or query value into the property, but it is also listening for changes to that property, which will then be written back to the tag or the database.

Tag Bindings

Tag bindings can be made bidirectional simply by checking the checkbox. The "Fallback Delay" is the amount of time that the value will remain at the written value, waiting for a tag change to come in. If no tag change comes in within the allotted time (specified in seconds), then the property will fall-back to the value as it was before the write. This is needed, because sometimes even if a write succeeds, another write or ladder logic in a PLC might have written something different, even the old value, in which case no tag change event will be generated. As a rule of thumb, the fallback delay should be twice the tag's scan class rate.

Query Bindings

When a query binding is made bidirectional, it needs an UPDATE query to execute when the property changes. You can use the special marker `{this}` as a placeholder for the new value.

Bidirectional query bindings are only available on scalar-typed properties (i.e. not Datasets)

4.7.4.4 Indirect Bindings

Making bindings indirect is an important part of the binding system. Indirect Tag, Expression, and SQL Query bindings can all be made indirect. All this means is that what the binding is bound to can be changed based upon the value of something else.

For example, instead of binding straight to a tag's path, like

```
[TagProvider]MyPlant/EastArea/Valves/Valve4/FlowRate
```

you can use other properties to make that path indirect. Suppose the "area" and valve number that we

© 2011 Inductive Automation

were looking at was passed into our window via parameter passing. Then we might use those parameters in the tag path, like this:

```
[TagProvider]MyPlant/{1}Area/Valves/Valve{2}/FlowRate
{1}=Root Container.AreaName
{2}=Root Container.ValveNumber
```

Now our binding will alter what tag it is pointing to based upon the values of those root container properties.

Making query bindings indirect, or dynamic, is so common that there are probably more indirect query bindings than direct ones. All this means is that the query is calculated dynamically. A common example of this would be to use a dynamic start date and end date in a query. Suppose we had a Classic Chart that we're binding to a range of history, and a Date Range that we wanted to have the operator use to select a time period. Then we could use an indirect query binding like this:

```
SELECT
   t_stamp, flow_rate, amps
FROM
   valve_history
WHERE
   t_stamp >= '{Root Container.DateRange.startDate}' AND
   t_stamp <= '{Root Container.DateRange.endDate}' AND
   valve = {Root Container.ValveNumber} AND
   area = '{Root Container.AreaName}Area'
```

See also:
Parameterized Windows

4.7.4.5 Binding Types

4.7.4.5.1 Tag Binding

A tag binding is a very straight-forward binding type. It simply binds to a tag property. This sets up a tag subscription for that tag, and every time the chosen property changes, the binding is evaluated, pushing the new value into the property.

If the tag is in a leased scanclass, this binding will activate the lease while the window is open.

If you choose a tag in the tree, and not a property, the `Value` property is assumed.

Bidirectional Mode
Choosing bidirectional will make this binding also write to the chosen tag when the property changes. The fallback delay is the amount of time to keep the property at the written value waiting for a new tag value update to come in. If no update arrives within the given timeout, the property falls back to the original value. See Bidirectional Bindings.

Overlay Opt-Out
Choosing this option will ignore the quality of the chosen tag, making it have no effect on the component's quality overlay.

© 2011 Inductive Automation

4.7.4.5.2 Indirect Tag Binding

An indirect tag binding is very much like a standard tag binding. except that you may introduce any number of indirection parameters into the path. These parameters are numbered starting at one, and denoted by braces, e.g. `{1}`.

The binding will be bound to the tag represented by the tag path after the indirection parameters have been replaced by the literal values they are bound to. An indirection parameter may represent a property on any component in the same window, or the value of any tag.

Indirect tag bindings can use bidirectional mode just like standard tag bindings.

4.7.4.5.3 SQLTags Historian Binding

This binding type (which is only available for Dataset type properties), will run a query against the SQLTags Historian.

Selected Historical Tags
For this type of query, you must select at least one tag path to query. The Dataset returned by the query will have a timestamp column, and then a column for each path that you select here.

These paths may use indirection following the same rules as the Indirect Tag Binding. Simply type the indirection parameters (e.g. {1}) into a selected tag path by double-clicking in the list of selected paths. All valid parameters will appear in the lower indirection table.

Date Range
Choose either a *Historical* or *Realtime* query. Historical queries use a date range that must be bound in from other components on the screen, typically a Date Range or a pair of Popup Calendars. Realtime queries always pull up a range that ends with the current time, so all they need is a length.

Sample Size and Aggregation Mode
The sample size determines how the query results will look. A *Natural* query will look up the logging rate for the queried tags, and return results spaced apart at that rate. This means that the return size will vary with the date range. An *On Change* query will return points as they were logged. This means that the results may not be evenly spaced. A *Fixed* query will return the given number of rows. Where data was sparse, interpolated values will be added. Where data is dense, the *Aggregation Mode* will come into play.

The *Min/Max* aggregation mode will return the min and max for every timestamp. The *Average* aggregation mode will return the average timestamp for data within the underlying range.

Return Format
Return format dictates how the requested data will be returned. The options are "wide" (default), in which each tag has its own column, and "tall", in which the tags are returned vertically in a "path, value, quality, timestamp" schema.

SQLTags Historian information is often easiest to work with in the Easy Chart component, which handles all of these options automatically.

See also:
How SQLTags Historian Works
Data Types
system.tag.queryTagHistory

© 2011 Inductive Automation

4.7.4.5.4 Property Binding

A property binding is a very simple type of binding. It simply binds one component's property to another. When that property changes, the new value is pushed into the property that the binding is set up on.

Why aren't all properties listed? You may notice that the list of properties available to bind *to* is smaller than the list of all properties. While nearly all properties can be *bound*, only some properties can be *bound to*. Only properties for which a `propertyChangeEvent` is fired may be bound to.

4.7.4.5.5 Expression Binding

An expression binding is one of the most powerful kinds of property bindings. It uses a simple expression language to calculate a value. This expression can involve lots of dynamic data, such as other properties, tag values, results of Python scripts, queries, etc.

Expressions can be used for many different purposes. Anytime information needs to be massaged, manipulated, extracted, combined, split, etc - think expressions.

Example
You have 3 bits in a PLC, only one of which will be on at a time. You want to turn these three bits into a single integer (0,1,2) to drive a component's Styles. Bind a dynamic integer property to:

```
binEnum({MyTags/Bit1}, {MyTags/Bit2}, {MyTags/Bit3})
```

Example
You have a Date, and need to extract the year, and concatenate the word "Vintage" to the end for a label display. Bind a label's `text` property to:

```
dateExtract({Root Container.VintageDate}, 'year') + ' Vintage'
```

Example
You have a button that starts a batch, but you only want to let it be pressed after the operator has entered a scale weight. Bind the button's `enabled` property to:

```
{Root Container.EntryArea.WeightBox.doubleValue} > 0.0
```

Example
You want to display a process's current state, translating a code from the PLC to a human-readable string, use of these two expressions (they're equivalent)

```
if ({CurrentProcessState} = 0, "Not Running",
if ({CurrentProcessState} = 1, "Warmup phase - please wait",
if ({CurrentProcessState} = 2, "Running", "UNKNOWN STATE")))
```

- or -

```
switch ({CurrentProcessState},
    0,1,2,
    "Not Running",
    "Warmup phase - please wait",
    "Running",
    "UNKNOWN STATE")
```

See also:
Expressions Overview

4.7.4.5.6 DB Browse Binding

This binding is technically equivalent to the SQL Query binding, except that it helps write the queries for you. Using the database browser, you can pick the table that you want to pull content from. If you have a fixed range of data to choose, simply select it in the table, and watch the query get generated.

© 2011 Inductive Automation

In the browse tree, you can choose which columns should act as your keys (these columns get put in the WHERE clause based on your selection) and which columns should be used to sort the data (these columns get put in the ORDER BY clause).

This binding type also serves as a convenient jumping-off point for the more flexible SQL Query binding. Construct the basic outline of your query in the DB Browse section, and then flip over to the SQL Query binding. Your query will be retained and can then be improved by hand.

4.7.4.5.7 SQL Query Binding

The SQL Query binding is a polling binding type that will run a SQL Query against any of the database connections configured in the Gateway.

Dynamic Queries

Using the brace notation, you can include the values of component properties (within the same window) and tag values inside your query. This is a very common technique to make your query dynamic. The values of the property or tag represented are simply substituted into the query where the braces are.

Note that because the substitution is direct, you'll often need to quote literal strings and dates to make your query valid. If you're getting errors running your query complaining about syntax, it is important to realize that these errors are coming from the database, not from Ignition. Try copying and pasting your query into the Query Browser and replacing the braces with literal values.

Example
A common requirement is to have a query filter its results for a date range. You can use the Date Range component or a pair of Popup Calendar components to let the user choose a range of dates. Then you can use these dates in your query like this:

```
SELECT
   t_stamp, flow_rate, amps
FROM
   valve_history
WHERE
   t_stamp >= '{Root Container.DateRange.startDate}' AND
   t_stamp <= '{Root Container.DateRange.endDate}'
```

Notice the single quotes around the braces. This is because when the query is run, the dates will be replaced with their literal evaluations. For example, the actual query sent to the database might look like this:

```
SELECT
   t_stamp, flow_rate, amps
FROM
   valve_history
WHERE
   t_stamp >= '2010-03-20 08:00:00' AND
   t_stamp <= '2010-03-20 13:00:00'
```

Fallback Value

If the property that is being bound is a *scalar* datatype (i.e. not a Dataset), then the value in the first column in the first row of the query results is used. If no rows were returned, the binding will cause an error unless the *Use Fallback Value* option is selected. The value entered in the fallback value text box will be used when the query returns no rows.

When binding a Dataset to a SQL Query, no fallback value is needed, because a Dataset will happily contain zero rows.

© 2011 Inductive Automation

See also:
Polling Options
Creating a Database Connection

4.7.4.5.8 Cell Update Binding

The Cell Update binding enables you to easily make one or more cells inside a dataset dynamic. This particularly useful for components such as the Linear Scale or the Easy Chart, that store configuration information inside datasets.

For example, when you configure *indicators* on a Linear Scale component using that component's customizer, the indicators that you set up are stored in the "Indicators" property on the scale. Suppose you wanted high-setpoint and low-setpoint indicators on the scale that weren't simply static values, but actually bound to a SQLTag indicating the realtime high and low setpoints. In order to do this, you'd set up a Cell Update binding on the Linear Scale's Indicators property. You would configure two cell bindings - one for the low setpoint indicator's Value column, and one for the high setpoint. You would then bind these to the appropriate tags.

As another example, let's say you had an Easy Chart on a window that displayed 5 pens representing the history of a Compressor: running status, amperage, rpm, output pressure etc. Using SQLTags Historian, you had simply dragged the 5 applicable tags onto the Easy Chart. But now you want to use that same Easy Chart to dynamically display the same 5 pens of *any* of the many compressors in your system. To do this, you could pass the compressor number into the window as a parameter, and use it to calculate the tag path of the folder containing the pens. Then set up a Cell Update binding on the Easy Chart's "Tag Pens" property, dynamically altering the pens' tag paths. Now you have a generic chart window that can be used for any compressor.

Note that this binding type is only applicable for Dataset-typed properties.

4.7.4.5.9 Function Binding

This is a generic binding type that allows you to bind a dataset property to the results of a function. It allows any of the function's parameters to be calculated dynamically via tag and property bindings. The function that you choose determines the parameters that are available.

4.7.5 Event Handlers

4.7.5.1 Overview

Event handling allows you to use scripting to respond to a wide variety of events that components fire. This lets you configure windows that are very interactive, and are an important part of project design in the Vision module.

Events

An event can be many things, like a mouse click, a key press, or simply a property changing. Whenever these events occur, a script can be called to "handle" the event. Different components can fire different types of events. For example, mouse events are very common and are fired by almost all components. The `cellEdited` event, on the other hand, is only fired by the Table component.

Configuring Handlers

To configure event handlers for a component, right click on it and choose the *Event Handlers...* item. You can also get to this button vial the toolbar ( ) or the *Component* menu. Once in the event handler window, you can pick any event to handle. Each event can have its own handling logic.

© 2011 Inductive Automation

Script Builders

All events are handled with scripting, but you frequently don't need to write the scripts by hand. This is where the *Script Builders* come in. For each event, you can choose a common way of handling the event. This can be a navigation action, setting a tag value, etc. To write an arbitrary script, choose the *Script Editor* tab.

For example, one of the most common uses of event handlers is to open a window when a button is pushed. To do this, simply select the `actionPerformed` event, and select the *Navigation* tab. Here you can simply pick the navigation action *Open*, and choose the window to open. If you're curious, you can peek over at the *Script Editor* tab to see the underlying code that makes this action tick, but you certainly don't have to.

See also:
About Scripting

4.7.5.2 The 'event' object

Event handling scripts are just regular Python scripts except for one important detail. They all have a special variable defined in their namespace called "`event`". This is an object that represents information about the event that just occurred. For example, the `event` object for a mouse click will have the x and y coordinates where the click occurred. A key press event, on the other hand, will have a keycode, but not a coordinate.

In addition to information about the event that has just occurred, the `event` object has a `source` property. The *source of an event is the component that fired it*. This is a crucial concept to understand. The reference to the component is your handle into the entire hierarchy of the window that your script is contained in.

Example
Suppose you're handling the mouse pressed event of a label component. The following script would print out the coordinates of the click, as well as the text of the label:

```
currentText = event.source.text
print 'Mouse clicked on label "%s" at %dx%d' % (currentText, event.x, event.y)
```

The output would look like this if the label's text was "this is my label":

```
Mouse clicked on label "this is my label" at 27x99
```

Using the `event` object to access the component hierarchy

Because `event.source` is the *component* that fired the event, you can use this reference to access the entire hierarchy of your window. This means you can access properties of any other component in the window. You just need to know how to navigate up and down the component tree.

To navigate *up* the component tree (going from a component to its *parent* container), simply use the `parent` property.
To navigate *down* the component tree (going from a container to one if its *children*) you use the `getComponent(name)` function.
To navigate *sideways* (getting a reference to a *sibling* component) you simply go up one level and then back down.

Example
Suppose the component hierarchy in our window looked like this:

```
Root Container
```

© 2011 Inductive Automation

```
HeaderLabel
StartButton
Options
   ProductCode
   BatchSize
PreviewTable
```

This window has a start button, a header, some options, and a preview table. Lets say that it is a window that lets the operator start a new batch. It has some options that are grouped into their own container. Lets say that the Root Container also has some parameters that our start button needs to know about.

The following table shows some script expressions and what they will evaluate to if you're writing an event handler for the StartButton component:

`event.source`
... the StartButton
`event.source.parent`
... the Root Container
`event.source.parent.MyProperty`
... the value of dynamic property "MyProperty" on the Root Container
`event.source.parent.getComponent("Options")`
... the Options container
`event.source.parent.getComponent("Options").getComponent("ProductCode").selectedValue`
... the selected value of the ProductCode dropdown component
`event.source.parent.getComponent("PreviewTable").selectedRow`
... the index of the selected row in the PreviewTable

There is one exception to the pattern of using `.parent` to go up the hierarchy and using `.getComponent(name)` to go down. The parent of a root container is *not* the window, and a reference to the window does not have a `.getComponent(name)` function. To get a reference to a window, simply use system.gui.getParentWindow with any component's `event` object as the parameter. Once you have a reference to a window, you can use its `.rootContainer` property to get to the root of the component hierarchy, and from here you can follow the rules laid out above.

See also:
Working with Components

4.7.5.3 Event Types

These are all of the event types that are fired by the various components in the Vision module. Events are organized into *event sets*. For example, the `mouse` event set includes `mouseClicked`, `mousePressed`, and `mouseReleased`. All of the events in an event set share the same properties for their `event` object.

Event Sets

`action`
`cell`
`focus`
`internalFrame`
`item`
`key`

© 2011 Inductive Automation

mouse
mouseMotion
paint
propertyChange

action Events

Events

actionPerformed

Properties in 'event'

source

The actionPerformed event is fired when an "action" occurs. What that "action" is depends on the component. The most common example is the Button component. You should always use the action event on a button instead of a mouse click, because it will be fired whenever the button is pressed, whether it is via the mouse or the keyboard (via a mnemonic shortcut or tabbing over to the button and pressing enter or space). The Timer component is another example of a component that fires an action event. In this case, the action is the timer firing.

cell Events

Events

cellEdited

Properties in 'event'

source

oldValue - the previous value in the cell

newValue - the newly entered value for the cell

row

column

Cell events are fired by a Table component that has editable columns. When a user edits a cell, this event will fire. The oldValue and newValue properties in the event can be used to determine what value the cell used to hold, and what new value the user has entered. The row and column properties, both integers, show what position in the table's data property the edit occurred at.

Example

Commonly, the event handler for a cell event will issue a SQL update query to persist changes to the table back to an external database. You can use the row to determine what the primary keys were for the row that was edited by looking at the table's data property. You can use the column index to find the column name of the edited column.

```
columnName = event.source.data.getColumnName(event.column)
primaryKeyValue = event.source.data.getValueAt(event.row, "keycolumn")
query = "UPDATE mytable SET %s=? WHERE keycolumn=?" % columnName
system.db.runPrepUpdate(query, [event.newValue, primaryKeyValue])
```

© 2011 Inductive Automation

`focus` Events

Events

`focusGained`

`focusLost`

Properties in '`event`'

`source`

`oppositeComponent` - the component that either gave up focus to this component, or took it away

Focus events are fired for components that can receive input focus. For both the focus gained and focus lost events, you can also access the "opposite" component. For a focus gain, this is the component that previously had the focus. For a focus lost event, the opposite component is the component that took the focus away.

You can programatically request that focus be given to a component by calling the function `requestFocusInWindow()` on that function. This function is actually defined by Java's JComponent class, from which all Vision components extend.

If you are trying to alter the focus from within a focus event handler, you must wrap your code in a call to system.util.invokeLater. This will let your focus change be processed *after* the current focus change event that is being processed has a chance to finish.

`internalFrame` Events

Events

`internalFrameActivated` - fired when the window becomes the focused window

`internalFrameClosed` - fired after the window is closed

`internalFrameClosing` - fired just before the window is closed

`internalFrameDeactivated` - fired when the window loses focus

`internalFrameOpened` - fired the *first* time a window is opened after *not* being in the cache

Properties in '`event`'

`source`

Internal frame events are fired by windows. (They are known as "internal frames" in the underlying Java windowing system that the Vision component uses). Note that the source of these events is the window itself. To get the root container of the window, use `event.source.rootContainer`, *not* `event.source.getComponent("Root Container")`.

The Activated/Deactivated events get fired when the component receives or loses input focus. The Activated event is a more reliable event to use as a window startup event than the Opened event, because the Opened event will *not* be called if the window was opened when it was already cached.

See also:

Window Cache Policies

`item` Events

Events

`itemStateChanged`

© 2011 Inductive Automation

Properties in 'event'

source

stateChange - a code that will be equal to either the SELECTED or DESELECTED constants.

SELECTED - a constant representing a selection event.

DESELECTED - a constant representing a deselection event.

The itemStateChanged event is used by components that choose between a *selected* or *deselected* state. For example, a Check Box or Radio Button. You can respond to this event to be notified when the state has changed (via any mechanism - click, keyboard, property bindings, etc). To check whether the event represents a selection or a deselection, you compare the event's stateChange property with the SELECTED or DESELECTED constants, like this;

```
if event.stateChange == event.SELECTED:
   print "Turned ON"
else:
   print "Turned OFF"
```

key Events

Events

keyPressed - fires when a key is pressed while the source component has input focus. Works for all keyboard keys.

keyReleased - fires when a key is released while the source component has input focus. Works for all keyboard keys.

keyTyped - fired when a *character* key is pressed and then released while a component has input focus.

Properties in 'event'

source

keyCode - an integer code representing the key that was pressed or released. Only valid on keyPressed and keyReleased events. See table below.

keyChar - a string that represents the character that was typed, if applicable (e.g. used for letters, but not an F-key). Only valid on keyTyped event.

keyLocation - the location of the key. E.g. to differentiate between left shift from right shift.

altDown - true (1) if the alt key was held down during this event, false (0) otherwise.

controlDown - true (1) if the control key was held down during this event, false (0) otherwise.

shiftDown - true (1) if the shift key was held down during this event, false (0) otherwise.

Key events are used to respond to keyboard input. They will only be fired on components that receive input focus. Handling key events often involves checking exactly what key was pressed. These events make a distinction between *character* keys (A,B,C...) and non-printable keys (F3, Esc, Enter). All keys will get keyPressed and keyReleased events, but only character keys will get keyTyped events. For keyTyped events, checking what key was pressed is relatively simple, you can simply do a comparison on keyChar, like event.keyChar == 'a'. For other keys, however, you need to compare the keyCode to a constant, enumerated below. These constants can be referenced through the event object itself, like: event.keyCode == event.VK_ENTER.

Key Code Constants

© 2011 Inductive Automation

VK 0 - VK 9	VK_END	VK_PAGE_UP
VK A - VK Z	VK_ENTER	VK_RIGHT
VK F1 - VK F24	VK_HOME	VK_SHIFT
VK ALT	VK_INSERT	VK_SPACE
VK CONTROL	VK_LEFT	VK_TAB
VK DOWN	VK_PAGE_DOWN	VK_UP

Location Code Constants

KEY_LOCATION_LEFT	KEY_LOCATION_RIGHT	KEY_LOCATION_UNKNOWN
KEY_LOCATION_NUMPAD	KEY_LOCATION_STANDARD	*(indeterminate or irrelevant)*

☞ All of this information comes straight out of the Java documentation for `java.awt.KeyEvent`. See http://java.sun.com/j2se/1.5.0/docs/api/java/awt/event/KeyEvent.html

`mouse` Events

Events

`mouseClicked` – fired when the mouse is pressed and released in the same spot on the component.

`mouseEntered` – fired when the mouse is moved so that it is hovering over the component

`mouseExited` – fired when the mouse had been hovering over the component and exits

`mousePressed` - fired when the mouse is pressed within the bounds of the component

`mouseReleased` - fired when the mouse is released after having been pressed within the bounds of the component

Properties in '`event`'

`source`

`button` – an integer code representing the button that was clicked. Use the constants `event.BUTTON1`, `event.BUTTON2`, and `event.BUTTON3`.

`clickCount` – an integer count of the number of successive clicks.

`x` – the x-axis location of the mouse click, with (0,0) being the upper left corner of the component.

`y` – the y-axis location of the mouse click, with (0,0) being the upper left corner of the component.

`popupTrigger` – true(1) if this mouse event should pop up a context menu. Meaning is OS-dependent. On windows, it is a release of `BUTTON3`.

`altDown` – true (1) if the alt key was held down during this event, false (0) otherwise.

`controlDown` – true (1) if the control key was held down during this event, false (0) otherwise.

`shiftDown` – true (1) if the shift key was held down during this event, false (0) otherwise.

`mouseMotion` Events

Events

`mouseDragged` – fires when the mouse is pressed within the component, and then moved. Will continue to fire until the button is released, even if the mouse moves outside the component.

`mouseMoved` – fired when the mouse moves over the component.

Properties in '`event`'

© 2011 Inductive Automation

see mouse events.

`paint` Events

Events

`repaint`

Properties in '`event`'

`source`

`graphics` - An instance of `java.awt.Graphics2D` that can be used to paint this component. The point (0,0) is located at the upper left of the component.

`width` - The width of the paintable area of the component. This takes into account the component's border.

`height` - The height of the paintable area of the component. This takes into account the component's border.

This event is fired by the Paintable Canvas component. This component is provided for highly script-literate users, and is decidedly *not* user-friendly. Don't say you weren't warned. It allows you to use Java2D through Python to programatically "paint" your own dynamic, vector-based component. This event is called every time the component needs to repaint. It will repaint when any of its dynamic properties change, or when `.repaint()` is called on it. Drop a Paintable Canvas onto a window and look at the paint event handler for an example.

`propertyChange` Events

Events

`propertyChange`

Properties in '`event`'

`source`

`newValue` - The new value of the property

`oldValue` - The previous value of the property. Not all properties provide this information.

`propertyName` - The name of the property that has changed.

The `propertyChange` event is called any time a *bindable* property changes on a component. This includes all dynamic properties. This can be a very useful tool, allowing you to respond via scripting when a property changes. Because this one event handler is called for multiple properties, it is typical for a handler to first have to filter based on the `propertyName`, so that it is responding to a specific property changing.

Example

```
#This script might go on a Table whose data must be filled in before continuing
if event.propertyName == "data":
   newData = system.db.toPyDataSet(event.newValue)
   if len(newData)>0:
      # Data exists - let the user know they may proceed
      system.gui.messageBox("You may proceed.")
```

© 2011 Inductive Automation

4.7.5.4 Script Builders

When creating an event handler, you can use one of the handy "script builders" instead of writing the script by hand. In the Event handlers configuration window, the script builders are accessible as tabs along the top. The last tab, "Script Editor", lets you write an event handler by hand. You can also use it to view the script that was generated by the script builder, which is a good way to get started learning how to write event handlers by hand.

Action Qualifiers
All of the script builders allow you to put security and/or confirmation qualifiers onto the event handler. The security qualifier lets you restrict the event handler from running if the current user doesn't possess a set of roles. Use CTRL-select to pick multiple roles. The confirmation qualified will prompt the user with a popup Yes/No box. The action will only be executed if the user chooses "Yes".

Navigation

The navigation script builder has various functions that deal with opening and closing windows.

Open / Swap
Opening is a very straight-forward operation - it simply opens the specified window. You are also given options to then center that window within the Client, and to close the window that the event was fired from.

S*wapping* is the practice of opening another window in the same size, location, and state as the current window, and closing the current window. This gives the appearance of one window simply swapping into another, seamlessly. The navigation builder uses the swapWindow version of swapping, but most "by hand" script authors will us the swapTo version. This last version relies on the fact that the windows being swapped are both maximized windows. See the typical navigation strategy section for more information.

You can also pass *parameters* to the opened or swapped-to window. The names of these parameters must match names of dynamic properties on the root container of the target window. The values can either be literals or values of other properties from the source window. To use a property, highlight an empty cell in the Value column of the parameter table, and press the Insert Property ( ) button. See the parameterized windows section for more information.

Forward / Back
These action give you a simple way of implementing "browser"-style forward/back buttons in your client. Note that you must be using the default navigation strategy for this to work, because these functions rely on calls to `system.nav.swapTo` in order to keep track of what the sequence of recent windows has been.

Closing Windows
These options allow for an easy way to have an event handler close the window that it is a part of, or any other window.

See also:
Parameterized Windows
Typical Navigation Strategy
system.nav.openWindow
system.nav.swapWindow

© 2011 Inductive Automation

Set Tag Value

This event handler script builder will respond to an event by setting the value of a SQLTag. You can set the tag to either a literal value directly typed in, or you can use the Insert Property ( ) button to have the handler use the value of another property from the same window.

SQL Update

This script builder helps you build an update query using a database browsing interface. Choose a spot in your target database and the update query will be built for you. By setting columns as key columns, you can have the filter correctly filter to the right row. You may use either literal values or property values by using the Insert Property ( ) button next to the *Update Value* text box.

Set Property

This script builder will respond to an event by altering a property in the window. You must choose the property to alter, and the value that you wish to assign to it. The value can be a literal value or the value of any other property on the window by using the Insert Property ( ) button.

4.7.6 Security

4.7.6.1 Role-based access

Security is configured using *roles*. This simple concept just means that instead of granting or revoking privilege based on *user*, you do so based upon the more abstract concept of a *role*, and then you assign users to belong to one or more roles.

The maintenance ramifications of this separation are fairly obvious - you define your security based upon the process, not the people. Ideally, the process remains constant even if the cast of characters changes. As people are hired, transferred, promoted, fired, etc, the security management simply becomes the re-assigning of roles, not the re-designing of your project.

Project Required Roles

The coarsest level of security for a Vision project is the project's *Required Roles* property. This is a list of roles that the user must have *all* of in order to log into the Client.

See also
Project General Properties
Gateway Configuration - Security Overview

4.7.6.2 Tag Security

SQLTags security is often the best way to configure security for data access. By defining security on a tag, you affect the tag across all windows in the project, as opposed to configuring component security on each component that displays or controls that tag.

If a user opens a window that has components that are bound to a tag that the user doesn't have clearance to read or write to, the component will get a *forbidden* overlay.

Tag security in action

© 2011 Inductive Automation

See also:
Quality Overlays
Tag Permissions

4.7.6.3 Component Security

Each window and component can define its own security settings. These settings determine who can see and/or use the component. To define security for a component, right click on it and choose "Component Security". Here you can choose to implement a security policy different than that of your parent.

In the Client, if the user does not match the role filter that you define, the component will be disabled or hidden and disabled. If a user with higher privileges logs in, the component will be useable again.

⚠ If you choose to *disable* a component, make sure that it is a component that actually *does something different* when it is disabled. For example, buttons and input boxes can't be used when they are disabled, but disabling a label has no effect.

4.7.6.4 Securing event handlers

Event handlers often execute logic that must be secured. The various script builders all have special security qualifiers that can be enabled. These qualifiers get translated into the generated script by accessing the user's current roles via scripting.

Example

```
if 'Administrator' in system.security.getRoles():
   productCode = event.source.productCode
   qty = event.source.parent.getComponent("QuantityBox").intValue
   query = "UPDATE my_secure_table SET quantity=? WHERE product=?"
   system.db.runPrepUpdate(query, [qty, productCode])
else:
   system.gui.errorBox('Insufficient security privileges.')
```

See also:
Script Builders
system.security.getRoles

© 2011 Inductive Automation

Scripting

Part V

inductive automation
800.266.7798
www.inductiveautomation.com

5 Scripting

5.1 About Scripting

Scripting is used in many places in Ignition to add a significant degree of flexibility and customization where pre-canned options fall short. There are two major scripting languages in Ignition, Python and the Expression Language. It is important to understand the differences between the two, and to know where each is used.

Python Scripting

What is Python?
Most of the time when we talk about "scripting" we're talking about Python scripting. Python is a general purpose programming language that was developed in the early 90's and has gained significant popularity in the 2000's. We like it because it is extremely readable, elegant, powerful, and easy to learn. As an added bonus, it gracefully interacts with Java, giving programmers an extremely powerful tool when paired with Ignition, which is written in Java.

Python or Jython?
You'll often hear Python referred to as "Jython" by advanced users of Ignition. Python is the language, Jython is the *implementation* of the language that we use. Most users of Python use the implementation called "CPython" - they just don't realize it. See http://en.wikipedia.org/wiki/Python_(programming_language)#Implementations

Why not VBA?
Many HMI/SCADA packages use VBA, or Visual Basic for Applications. As such, many engineers switching to our software inquire about it. There are a variety of reasons we don't use VBA:

1. It is not compatible with Java, the language that Ignition is written in. This also means that it is not cross-platform.
2. It is a dying language (Microsoft is phasing it out as of July, 2007)
3. It is full of security holes
4. It is an ugly language

Where is Python Used?
Python is used in many places in Ignition. The most apparent place is in component event handlers. Project event scripts are another major place where Python is used.

Expression Language

The expression language is a simple language that we invented. An expression language is a very simple kind of language where everything is an *expression* - which is a piece of code that returns a value. This means that there are no *statements*, and no *variables* , just *operators, literals,* and *functions.* The most common expression language that most people are familiar with is the one found in Excel. You can have Excel calculated a cell's value dynamically by typing an expression like `=SUM(C5:C10)`. Our expression language is similar. It is used to define dynamic values for tags and component properties.

5.2 Python

5.2.1 About Python

While it is entirely possible to create a complete and powerful project in Ignition without writing a line of script, many designers will find that in order to complete projects with specific requirements, they need to learn at least a little Python. In our experience, most industrial projects involve lots of very complex

© 2011 Inductive Automation

and specific requirements.

The good news is that learning Python is easy and enjoyable. Python is one of the most beautiful programming languages we've ever encountered. It is very easy to read - even if you don't know it at all, you will probably be able to understand a basic Python script. It is frequently called it "executable pseudocode". We've included a short tutorial here which should help get you started. If you find yourself doing a lot of scripting, you may want to pick up a basic reference book about Python.

Scripting Help

Scripting is one of the topics in Ignition that users frequently need help with, because it is used to achieve some of the most complex requirements of a project. If you get stuck designing a script, or would like help getting started, don't hesitate to get some help. Our user forum at http://www.inductiveautomation.com/forum is by far the best place for scripting help.

When asking for scripting help - be precise and complete. If you're working through an error - include the text of the error, the circumstances, and the offending code. If you're stuck on something, it is helpful to describe the broader goals of what you're trying to accomplish - there is often an easy way and a hard way. Don't be shy to simply ask for some direction getting started.

Under the hood - Python in Java

The implementation of Python included in Ignition is Jython 2.1. Every once in a while we get asked if we have plans to "upgrade" our underlying scripting engine to a more recent version. The answer is yes, but it is a fairly large project to do and still maintain backwards compatibility, which is our first priority.

One of the powerful things about using Jython is that your script has access to the entire Java standard library. In the Client, this will be Java 5 or Java 6. When running under the Gateway, this will always be Java 6. See Accessing Java for more information.

Many scripting users are blown away by their script's speed. We can't take credit for this - the Jython engine hot-compiles your Jython code to Java bytecode, which means it runs natively in the JVM, which in turn can hot-compile it to machine code. It's fast.

5.2.2 Python Tutorial

5.2.2.1 Basic Syntax

The basic syntax of python is easy to learn, because there isn't much of it.

Hello World

Lets get right to everyone's favorite example: the following script will print out "`Hello World`" to the output console.

```
print "Hello World"
```

The `print` keyword is a handy tool in Python, allowing you to put text into the output console. This is useful for debugging your scripts. You can print multiple things by separating them with commas.

Variables

Variables are created by simply assigning a value to them. Variables do not need to be declared, because Python has a *dynamic type system*. That means Python figures out the type of the variable on the fly, when the script is executed.

The following script would print out: `15`

```
x=5
```

© 2011 Inductive Automation

```
y=3
print x*y
```

Strings, Numbers, and Booleans

Literal strings can be typed in using either double quotes or single quotes. This can be handy when your string contains one quote or the other. You can also use the backslash character to escape special characters.

Numbers can just be typed in normally, like `42` or `3.14159`. Python does not have a boolean type. 0 is false and 1 is true. (this is an oversimplification, but should suffice for now). The following prints out "`1`"

```
x="isn't this grand"
y='isn\'t this grand
print x==y
```

The None Value

There is a special value in Python called `None` (with a capital N). This is simply a special value that means: no value. This value is equivalent to Java's `null` value.

Lists

In Python, lists (arrays) are a built-in type that contains multiple other values. Lists can contain any type of items, and the items in a list do not all need to be the same type. You can create a list by enclosing multiple items in square brackets (`[]`), separated with commas. You can pull items out of a list with the square-bracket list index notation. Note that lists are zero-indexed, meaning that the first item in the list is at position 0. This code will print out "`a list`".

```
a = ['this', 'is', 'a list', 8, 93.928]
print a[2]
```

Basic operators

Python has all of the normal **arithmetic operators** you'd expect, addition(`+`), subtraction(`-`), division(`/`), multiplication(`*`), modulus(`%`), etc.

The comparison operators are just like in C: equals(`==`), not equals(`!=`) greater than (`>`), greater than or equal(`>=`), etc.

The **logical operators** are just typed in plain text: `and`, `or`, `not`.

These are just the basics. There are other operators, like bit shift operators. Read about them at: http://docs.python.org/library/stdtypes.html

Comments

Comments start with a hash sign. Add comments to your code so that when you go back to it after a long time, you know what the code is trying to do.

```
# Prints out 'Hello World' 5 times.
for x in range(5):
   print 'Hello world'
```

Whitespace

Perhaps its most unique feature, logical blocks are defined by *indentation* in Python. A colon (`:`) starts a new block, and the next line must be indented (typically using a tab of 4 spaces). The block ends when the indentation level returns to the previous level. For example, the following will print out "`5 4 3 2 1 Blast-off`". The final print is *not* part of the loop, because it isn't indented.

© 2011 Inductive Automation

```
countdown=5
while countdown > 0:
   print countdown,
   countdown = countdown - 1
print "Blast-off!"
```

5.2.2.2 Control Flow

Control flow are the parts of a language that make it do things differently based upon various conditions. In other words: **if**s and **loops**. Python has all of the basic control flow statements that you'd expect.

`if` Statements

If statement should be familiar to anyone with a passing knowledge of programming. The idea of an `if` is that you want your script to execute a block of statements only if a certain condition is true. For example, this script won't do anything.

```
x = 15
if x < 10:
   print "this will never show"
```

You can use the `if...else` form of an if statement to do one thing if a condition is true, and something else if the condition is false. This script will print out "`this will show!`"

```
x = 15
if x < 10:
   print "this will never show"
else:
   print "this will show!"
```

Lastly, you can use the `if...elif` form. This form combines multiple condition checks. "elif" stands for "else if". This form can optionally have a catch-all "else" clause at the end. For example, this script will print out "three":

```
x = 3
if x == 1:
   print "one"
elif x == 2:
   print "two"
elif x == 3:
   print "three"
else:
   print "not 1-3"
```

`while` Loops

A while loop will repeat a block of statements while a condition is true. This code will print out the contents of the items in the list. This code uses a *function* called `len`, which is a built-in function that returns the length of a list or string.

```
listOfFruit = ['Apples', 'Oranges', 'Bananas']
x = 0
while x < len(listOfFruit):
   print listOfFruit[x]
   x = x + 1
```

`for` Loops

Python's for loop may be a bit different than what you're used to if you've programmed any C. The for loop is specialized to iterate over the elements of any sequence, like a list. So, we could re-write the example above using a for loop eliminating the counter x:

```
listOfFruit = ['Apples', 'Oranges', 'Bananas']
```

© 2011 Inductive Automation

```
for item in listOfFruit:
   print item
```

Much more graceful! You'll often see the for loop used instead of the while loop, even when you simply want to iterate a given number of times. To do this with the for loop, you can use the built-in function `range`. The `range` function returns a variable-size list of integers starting at zero. Calling `range(4)` will return the list `[0, 1, 2, 3]`. So, to have a for loop repeat 4 times, you simply can do:

```
for x in range(4):
   print "this will print 4 times"
```

`break` and `continue` in Loops

You can stop a loop from repeating in its tracks by using the `break` statement. This code will print out "`Loop`" exactly *two* times, and then print "`Finished`".

```
for x in range(10):
   if x >= 2:
      break
   print "Loop"
print "Finished"
```

You can use the `continue` statement to make a loop stop executing its current iteration and skip to the next one. The following code will print out the numbers 0-9, skipping 4

```
for x in range(10):
   if x == 4:
      continue
   print x
```

5.2.2.3 String Formatting

String formatting is a somewhat minor feature of Python, but turns out to be incredibly useful in Ignition. String formatting is used to manipulate strings, specifically to insert the values of variables inside a string without a bunch of concatenation.

The `%` operator is used in Python not just for modulus, but also for string formatting. Suppose we wanted to print a weather report. We could use concatenation, like this:

```
temp = 65.8
city = "Sacramento"
windSpeed = 25
windDir = "east"

print city + " weather: " + str(temp)  + "°F, wind "+ str(windSpeed) + "mph from the "+ windDir
```

Yuck! This kind of concatenation is really a pain to write and to read. With string formatting, we could have written it like this:

```
temp = 65.8
city = "Sacramento"
windSpeed = 25
windDir = "east"

print "%s weather: %f°F, wind %dmph from the %s" % (city, temp, windSpeed, windDir)
```

Ah, that's much easier on the eyes. What is happening here is that the `%` operator is applying the variables on its right-hand side to the *format string* on its left-hand side. It looks for placeholders (called *format specifiers*) inside the format string, and replaces them with corresponding values from the

© 2011 Inductive Automation

variables on the right-hand side. There are various format specifiers that can be used for different types of variable types. If you actually want a % sign inside the final string, use the special format specifier: "%%"

Format Specifier	Meaning
%%	Inserts a % sign into the final string
%c	A single character. Value must be a string of length 1 or an integer
%d or %i	Signed integer
%f	Floating point, decimal format
%s	A String, converts the value to a string using str()
%u	Unsigned decimal
%x or %X	Unsigned hexadecimal

You can also put some extra information in the format specifiers between the % and the format specifier character. The most useful thing to do is to specify the number of decimal places to use to print floating point numbers. For example, "%.3f" would always put three digits after the decimal point.

5.2.2.4 Functions

Functions are code that can be called repeatedly from other places. Functions can have parameters passed into them, and may return a resulting value. Some functions, like `len`, are built-in. Some functions, like `system.gui.messageBox()`, are part of the scripting libraries provided by Ignition. Some functions, like `math.sqrt()`, are provided by the Python standard libraray.

Functions are `invoked` by using their name followed by an argument list surrounded in parentheses. If there are no arguments, you still need an open and close parenthesis.

Defining Functions

Functions are defined using the `def` keyword. A function needs a name, and needs a list of the arguments that it can be passed. For example, this code defines a function that tests whether or not a number is odd. It returns a true value (1) if the number is odd. It is then used in a loop to print out the odd numbers between 0 and 9.

```
def isOdd(num):
   return num % 2 == 1 # uses the modulus (or remainder) operator

for x in range(10):
   if isOdd(x):
      print x
```

Function Arguments

When a function accepts arguments, the names of those arguments become variables in the function's namespace. Whatever value was passed to the function when it was invoked becomes the value of those variables. In the example above, the value of `x` inside the `for` loop gets passed to the `isOdd` function, and becomes the value of the `num` argument.

Arguments can have *default values*, which makes them optional. If an argument is omitted, then its default value will be used. The following code defines a function called `cap`, which will take a number, and make sure it is within an upper and lower limit. The limits default to 0 and 100.

```
def cap(x, min=0, max=100):
   if x < min:
      return min
   elif x > max:
      return max
```

© 2011 Inductive Automation

```
   else:
      return x

# This will print out "0"
print cap(-1)

# This will print out "100"
print cap(150)

# this will print out "150", because it uses a max of 200
print cap(150, 0, 200)
```

Keyword Arguments

Arguments can also be specified by *keyword* instead of by position. In the above example, the only way someone would know that the 200 in the last call to `cap` specified the max is by its position. This can lead to hard-to-read function invocations for functions with lots of optional arguments. You can use keyword-style invocation to improve readability. The following code is equivalent to the last line above, using 200 for the `max` and the default for the `min`.

```
print cap(150, max=200)
```

Because we used a keyword to specify that 200 was the "max", we were able to omit the min argument altogether, using its default.

Note that not all functions in the standard library and the Ignition library can be called with keyword invocation. Functions that accept keyword invocation, like system.tag.queryTagHistory, will say so in their documentation.

Functions are Objects

Perhaps one of the most foreign concepts for new Python users is that in Python, functions are first-class objects. This means that functions can be passed around to other functions (this concept is similar to the idea of *function pointers* in C or C++).

Lets go back to the `isOdd` example above. Suppose we wanted a more general way to filter a list. Maybe sometimes we want the odd entries, while other times we want even ones, or entries less than 3, etc. We can define a function called `extract` that takes a list and *another function*, and returns only entries that "pass" through the other function.

```
def isOdd(num):
   return num % 2 == 1

def isEven(num):
   return num % 2 == 0

def isLessThan(num, max=3):
   return num < max

def extract(filterFunction, list):
   newList = []
   for entry in list:
      if filterFunction(entry):
         newList.append(entry)
   return newList

# prints out [0, 2, 4, 6, 8]
# notice that isEven as not _invoked_, but passed to the filter function
print extract(isEven, range(10))
```

© 2011 Inductive Automation

Now, it just so happens that Python has a built-in function that does exactly what our `extract` function does - its called `filter`.

We would also be remiss at this point if we didn't mention another language feature called *list comprehensions*. This is a great little bit of syntax that helps make new lists out of other lists. Instead of using our filter function, we could have simply done this:

```
def isEven(num):
   return num % 2 == 0

print [x for x in range(10) if isEven(x)]
```

If that looks cool to you - read more about list comprehensions at http://docs.python.org/tutorial/datastructures.html#list-comprehensions

In Ignition, you'll most commonly see functions used as objects when using the system.util.invokelater function. This function takes a function and executes it after all pending event handling has finished processing.

5.2.2.5 Scope and Import

The concept of *scope* is very important in all programming, and Python is no exception. Scope defines what names are directly accessible without any qualifiers. Another way to put this is that the scope determines what variables are defined. When you define a new function, that function gets its own scope. The statements within the function don't operate in the scope of the enclosing code. An example should make this clear:

```
x = 5
print x
def fun():
   x = 3 # this 'x' is not the same as the outer 'x'
   print x

fun()
print x
```

This code will print:

```
5
3
5
```

The assignment `x = 3` within the function did not affect the `x` defined outside the function's scope. Furthermore, if you tried to access `x` within the function `fun` without the `x = 3` line, you would receive a `NameError`, because `x` would not be defined.

Global Scope

Besides your immediate scope, there is also the global scope. By declaring a name preceded with the keyword `global`, your variable will be resolved using the global scope, which is shared by all scripts.

```
global x
# will print whatever value some other script
# assigned to x in the global namespace
print x
```

© 2011 Inductive Automation

Using the `import` keyword

You can import the namespaces defined in other scopes into your scope with the `import` keyword. Most commonly, you'll import from global library sources, like `system` (the Ignition standard libraries), `app` (your project's global script modules), `java` (importing from the Java standard library), and the various python standard libraries. When you're writing component event handlers, system and app are imported for you automatically. However, if you create a new scope by defining a function, you'll need to import those libraries manually.

The import keyword can be used in a variety of forms:

```
import X
from X import *
from X import Y
```

For example, suppose you wanted to use the java.util.Calendar class for some date manipulations. You could import this in a number of different ways. These examples are equivalent, printing out a date 8 hours before the current date.

```
import java
cal = java.util.Calendar.getInstance()
cal.add(java.util.Calendar.HOUR, -8)
print cal.getTime()
```

```
from java.util import Calendar
cal = Calendar.getInstance()
cal.add(Calendar.HOUR, -8)
print cal.getTime()
```

5.2.2.6 Sequences and Dictionaries

Python offers a variety of *sequence* types. We've already seen one - the List. There are other kinds of sequences, most notably *tuples* and *strings*. There is also the *dictionary* type, which contains a list of key-value pairs.

Lists

Lists are a very handy kind of sequence. They can hold any number of items, can be resized on the fly. After creating a list using the square bracket notation, there are a number of functions that you can call on the list. Some common list functions are listed here. Visit http://docs.python.org/tutorial/datastructures.html#more-on-lists for a complete list.

`append(x)` - takes a single argument, which will be appended to the end of the list.
`insert(i,x)` - inserts an item x at index i
`remove(x)` - will remove the given item from the list.
`index(x)` - returns the index of the value x. Throws an error if the list doesn't contain the item. Use the `in` operator to check if an item is contained in a sequence.
`sort()` - sorts the items in the list.

```
myList = ['a', 'b', 'c', 'd']
print myList # --> [a, b, c, d]

myList.append("Q")
print myList # --> [a, b, c, d, Q]
```

© 2011 Inductive Automation

```
myList.insert(1, "Z")
print myList # --> [a, Z, b, c, d, Q]

myList.remove("c")
print myList # --> [a, Z, b, d, Q]

print myList[2] # --> b
print myLIst.index("b") # --> 2

if 'Z' in myList:
   print 'Z is in the list'

if 'c' not in myList:
   print 'c was removed from the list'
```

Tuples

A tuple is similar to a list, but you use parenthesis instead of square brackets to define one. The major difference between a tuple and a list is that tuple's are *immutable*. That is, once created, they cannot be altered. Tuples are very useful for passing multiple things to and from functions. For example, you could pass a point to a function using a tuple like so:

```
def printPoint(point):
   print "x = ", point[0]
   print "y = ", point[1]

printPoint((28,89))
```

This can also be handy for returning multiple values from a function. For example, if you had a mouse event, you could write a function that found the component's center point, and return that point as a tuple. You could then use *unpacking assignment* to extract the values into separate values.

```
def findCenter(event):
   w = event.source.width
   h = event.source.height
   return (w/2, h/2)

# point will be a tuple
point = findCenter(event)

# x and y will be numbers, using unpacking assignment
x,y = findCenter(event)
```

Dictionaries

A dictionary is a very useful type that holds a set of key-value pairs. You may have used these in other languages and know them as hashmaps, maps, associative memories or associative arrays. Dictionaries are not ordered sequences - you reference any item via its *key value*. The keys can be numbers, strings, or tuples of these types. Any given key may only appear once in a dictionary. Trying to set another value for that key will overwrite any previous value for that key.

Dictionaries are created using braces ({}). Key-value pairs are separated by commas, and the keys are separated from the values with a colon. You can use the .keys() function to have a set of the keys. For example:

```
myDict = {'Bob': 89.9, 'Joe': 188.72, 'Sally': 21.44}
```

© 2011 Inductive Automation

```
print myDict['Bob'] # --> 89.9

myDict['Amir']=45.89 # Adds a key for 'Amir'

names = myDict.keys()
names.sort()
print names # --> ['Amir', 'Bob', 'Joe', 'Sally']
```

You can loop through dictionaries using a for loop. You can use the `keys()` to loop through the dictionary, and then use the key values to look up the value. For example:

```
for name in myDict.keys():
   print name, myDict[name]
```

5.2.2.7 Exception Handling

Exception handling is a language feature of many high-level languages that allows you to "catch" a runtime error and deal with it as you see fit. On the flip side, it allows you to "raise" or "throw" an error in your code, which will break out of whatever code is currently executing and jump to the nearest enclosing catch block that knows how to handle your error.

For example, dividing by zero raises a ZeroDivisionError. You can catch this error using a `try... except` block, like this:

```
try:
   result = 8 / 0
   print "this will never get called"
except ZeroDivisionError:
   print "oops - can't divide by zero"
```

You don't have to specify a particular type of error to catch - you can use the except keyword by itself to catch any kind of exception. You can also assign the details of the exception to a tuple of variables, which you can use in your error reporting. You can also have multiple except blocks for one try block, each that handle different kinds of exceptions. This example shows these variations:

```
def someDangerousFunction():
   raise IOError(42,"oh no")
try:
   someDangerousFunction()
except IOError, (errno, description):
   print "An I/O error occurred: "+description
except:
   print "An unexpected error occurred"
```

You can learn more about exceptions at http://www.python.org/doc/2.1/tut/node10.html.

5.2.2.8 Learn More

Online Tutorials

Since Python is such a popular and well-regarded language, there are many high-quality tutorials available on the web. The official python tutorial, written by the inventor of Python himself, Guido van Rossum, is very good.
http://www.python.org/doc/2.1/tut/tut.html

The Non-Programmers Tutorial For Python by Josh Cogliati is also very good for those with no previous programming experience.
http://www.oopweb.com/Python/Documents/easytut/VolumeFrames.html

© 2011 Inductive Automation

You can go and download a printable Python "cheat sheet" from the Added Bytes website, available here:
http://www.addedbytes.com/cheat-sheets/python-cheat-sheet/

Recommended Books

Sometimes a good reference book is invaluable. The following books have gotten good reviews from us and our customers:

Learning Python (O'Reilly, 2007)
Python Pocket Reference (O'Reilly, 2005)
Core Python Programming (Prentice Hall, 2006)
Python Power (Course Technology, 2007)

Using Java

This book would be useful for anyone who finds themselves accessing the Java standard library frequently from Python:
Python Programming with the Java(TM) Class Libraries (Addison-Wesley, 2002)

You can also find the excellent API documentation for the Java standard libraries from Sun here:
http://java.sun.com/j2se/1.5.0/docs/api/index.html

Online Forum

Our online forum at http://www.inductiveautomation.com/forum is a great place to go for scripting help. Not only do we, the Inductive Automation staff, monitor it actively, but we have a thriving user community who can help you with any scripting questions.

5.2.3 Python in Ignition

5.2.3.1 Working with Different Datatypes

You'll encounter lots of different datatypes when scripting in Python. This guide should help you through the snags of working with some of the more complex types.

Numbers

Working with numbers is very easy in Python, and requires no special considerations. You can use the built-in function `int()` to attempt to coerce values to integers, and `float()` to coerce values to floating-point values. Both will throw `ValueError` if the coercion fails.

If you are new to programming, the following might throw you off. Python, like nearly all programming languages, uses *integer division* when dividing two integers. This means that `1/2` will result in `0`. This is because both 1 and 2 are integers, so the result of the division must be an integer. The result of 0.5 gets truncated to 0. If either operand is a float, the result will be a float, so `1 / 2.0` will result in `0.5`.

Strings

Strings are used frequently in scripting. Strings can be defined using double quotes *or* single quotes. Learning how to use String Formatting is a very useful technique. You can user the built-in function `str()` to coerce values into strings.

Colors

© 2011 Inductive Automation

Working with colors in Python is remarkably easy. You can simply use any tuple of 3 or 4 integers to represent a color in RGB or RGBA. For example, to set a label's text color to red, you can simple do something like this:

```
label = event.source
label.foreground = (255,0,0)
```

Dates

Dates are one of the trickier datatypes to deal with in scripting. It turns out that it is easier to deal with dates using the Java classes `java.util.Date` and `java.util.Calendar` than it is to use Python's `time` module.

Creating Dates

To create an arbitrary date, you can use the `java.util.Calendar` class. It has various functions to alter the calendar fields, like Calendar.HOUR, Calendar.MONTH, etc. After you're done manipulating the Calendar, you can use its `getTime()` function to retrieve the Date represented by the calendar. It also has a handy set() function that takes the common parameters of a Date. The one **major gotcha** here is that January is month *zero*, not month one. For example:

```
from java.util import Calendar
cal = Calendar.getInstance()

# set year, month, day, hour, minute, second in one call
# This sets it to Feb 25th, 1:05:00 PM, 2010
cal.set(2010, 1, 25, 13, 5, 0)
myDate = cal.getTime()
```

Date Arithmetic

Often you'll have a Date object from a component like the Popup Calendar and want to alter it programmatically. Say, subtracting 8 hours from it, or something like that. The `java.util.Calendar` class is used for this as well. Following the example above, this code would subtract 8 hours from the variable `myDate`.

```
from java.util import Calendar
cal = Calendar.getInstance()

cal.setTime(myDate)
cal.add(Calendar.HOUR, -8)
myNewDate = cal.getTime()
```

Date Formatting

To format a date as a String, you can use the system function system.db.dateFormat. This function uses a *format string* to give it a hint as to how you want your date formatted. The format string is full of various placeholders that will display different parts of the date. These are case-sensitive! The most common placeholders are:

y	Year
M	Month
d	Day
E	Day of Week
a	am/pm marker
H	Hour of day (0-23)
h	Hour in am/pm (1-12)
m	Minute
s	Second

© 2011 Inductive Automation

S	Millisecond
z	Time zone

These placeholders can be repeated for a different effect. For example, `M` will give you 1-12, `MM` will give you 01-12, `MMM` will give you Jan-Dec, `MMMM` will give you January-December. Here are some examples:

```
from java.util import Date

now = Date() # creates a new date, for right now

# Common format for databases
print system.db.dateFormat(now, "yyyy-MM-dd H:mm:ss")

# Nice human-readable format for just the date
print system.db.dateFormat(now, "MMM d, yyyy")

# Formating just the time in am/pm style
print system.db.dateFormat("h:mm a")
```

Datasets

It is very common to deal with datasets in scripting, as datasets power many of the interesting features in Ignition, like charts and tables. The `system.dataset` library provides various functions for manipulating and creating datasets.

The main confusion when dealing with datasets is the difference between the DataSet object and the PyDataSet object. DataSet is the kind of object that Ignition uses internally to represents datasets. When you get the `data` property out of a Table, for example, you'll get a DataSet. The PyDataSet is a *wrapper* type that you can use to make DataSets more accessible in Python. You can convert between the two with system.dataset.toPyDataSet and system.dataset.toDataSet.

Accessing data in a DataSet

DataSets have various properties and functions that you can access through Python.

`rowCount` - returns the number of rows in the dataset
`columnCount` - returns the number of columns in the dataset
`getColumnName(index)` - returns the name of the column at the given index
`getValueAt(row, column)` - returns the value from the dataset at the given location. column can be either an integer or a column name, which is treated case-insensitive.

For example, you could iterate through every item in a DataSet in scripting like this:

```
# Pull the dataset property off a Table component
data = event.source.getComponent("Table").data

for row in range(data.rowCount):
   for col in range(data.columnCount):
      print data.getValueAt(row, col)
```

or you could find specific values from each row in a DataSet like this:

```
# Pull the dataset property off a Table component
data = event.source.getComponent("Table").data

for row in range(data.rowCount):
   temp = data.getValueAt(row, "Temperature")
   speed = data.getValueAt(row, "Speed")
   print temp, speed
```

© 2011 Inductive Automation

Accessing data in a PyDataSet

You can convert a dataset to a PyDataSet, which lets you use it more like a Python sequence. You don't have to do this, its purely a convenience. A PyDataSet is like a list of dictionaries, and so it can use the normal for loop syntax. These examples are equivalent to the examples above.

Iterating through a PyDataSet

```
# Pull the dataset property off a Table component
data = event.source.getComponent("Table").data

# Convert to a PyDataSet
pds = system.dataset.toPyDataSet(data)

for row in pds:
   for value in row:
      print value
```

Finding specific values in a PyDataSet

```
# Pull the dataset property off a Table component
data = event.source.getComponent("Table").data

# Convert to a PyDataSet
pds = system.dataset.toPyDataSet(data)

for row in pds:
   temp = row["Temperature"]
   speed = row["Speed"]
   print temp, speed
```

Altering Datasets

Technically, you cannot alter a dataset. Datasets are *immutable*, meaning they cannot change. You can, however, create *new* datasets. So to alter a dataset, you really create a new one and then replace the old one with the new one. Because this is so common, there are special functions under system.dataset that are designed for this. You can use the following functions to create datasets that are altered version of existing datasets:

- system.dataset.addRow
- system.dataset.deleteRow
- system.dataset.setValue
- system.dataset.updateRow

The important thing to realize about all of these datasets is that, again, they do not actually alter the input dataset. They return a new dataset. You need to actually use that returned dataset to do anything useful. For example, this code would set the "Quantity" column in the selected row of a table to 15.8:

```
table = event.source.parent.getComponent("Table")
selRow = table.selectedRow
if selRow != -1:
   # Create a new dataset
   newData = system.dataset.setValue(table.data, selRow, "Quantity", 15.8)

   # Replace the Table's data property with the new dataset
   table.data = newData
```

Creating Datasets

© 2011 Inductive Automation

Sometimes you'll want to create a new dataset from scratch. This can be easily done with the system.dataset.toDataSet function. All it needs are the column headers and a list of the rows in the dataset. Each row must have the same number of elements as the header list. For example, this code would create a dataset that contained some information about US cities:

```
headers = ["City", "Population", "Timezone", "GMTOffset"]
data = []

data.append(["New York", 8363710, "EST", -5])
data.append(["Los Angeles", 3833995, "PST", -8])
data.append(["Chicago", 2853114, "CST", -6])
data.append(["Houston", 2242193, "CST", -6])
data.append(["Phoenix", 1567924, "MST", -7])

cities = system.dataset.toDataSet(headers, data)
```

5.2.3.2 Component Event Handlers

Using scripts to handle component events is one of the most common places to use scripting in Ignition. When an event occurs for a component, like a mouse click or a key press, you can have your script (the *event handler*) be called.

When your event handler is executed, it already has three names in scope:
`event` - the event object
`system` - the root of the Ignition Scripting API
`app` - the root of your project's script modules

See also:
Event Handlers Overview

5.2.3.3 Working with Components

When you're writing component event handlers, you'll do a lot of work with components. You'll need to reference various components on the window or on other windows, you'll need to reference and set properties of the component, you may even want to move components around on the screen.

Finding Components

When you have an `event` object, that object becomes your window into the entire component hierarchy. `event.source` references the component that fired whatever event you're responding to. `event.source.parent` references the container that component is in. `event.source.parent.getComponent("Name")` finds a sibling component with a certain name. The manual page for the event object covers this topic in more detail.

Accessing Component Properties

Once you have a reference to a component, you can treat it just like any Python object. You can call functions on it, and you can reference its properties, both standard and dynamic, with the "." accessor. For example, you could put this in a button next to the table, which would tell the user which row was selected, then clear the selection, and then print the table.

```
table = event.source.parent.getComponent("Table")

# Referencing properties of a component
row = table.selectedRow
system.gui.messageBox("The selected row is : %d" % row)
```

© 2011 Inductive Automation

```
# Setting properties of a component.
table.selectedRow = -1

# Calling functions on components
table.print()
```

Finding Components on Other Windows

Sometimes you may want to reference components on other windows. Or maybe you don't have an 'event' object because you're writing a project event script. In this case, you'll need to look up the containing window first. You can do this with the system.gui.getWindow function. This function will throw a ValueError if the given window isn't open, so you should make sure your code handles that gracefully. Once you have a Window, you can use its `rootContainer` property to get into the standard component hierarchy. This code will look up the HeaderLabel on a window named "Overview" and set its text and foreground color.

```
try:
   window = system.gui.getWindow("Overview")
   label = window.rootContainer.getComponent("HeaderLabel")
   label.text = "Notice Me!"
   label.foreground = (255,0,0)
except ValueError:
   # ignore error with a pass keyword
   pass
```

Common Component Functions

There are a number of functions that are common to all components in Ignition.

`requestFocusInWindow()` - requests that the component be given input focus. See also: Focus Events.
`setPropertyValue(name, value)` - sets the value of a component's dynamic property.
`getPropertyValue(name)` - gets the value of a dynamic property.

Moving/Resizing Components and Windows

You can use scripting to move and resize a component at runtime. The functions system.gui.moveComponent, system.gui.reshapeComponent and system.gui.resizeComponent are used for this. They simply take a component, and a new size, location, or both. Locations are always measured in pixels from the upper left point of the component's parent container.

Note that if you're moving a component that is set to relative layout, your coordinates will act as if they were coordinates to the sizes of the relevant containers *last time they were saved in the Designer*, not the current real coordinates of the runtime. This is very helpful for creating animations. In effect what this means is that the coordinates fed into these functions "respect" the relative layout system automatically.

You can move and resize windows as well. If you have a reference to a window, you can set its size and location directly. For example, if you wanted to move the floating window Popup3 to certain location, you could do so like this:

```
try:
   window = system.gui.getWindow("Popup3")
   window.setSize(250,600)
```

© 2011 Inductive Automation

```
   window.setLocation(0,0)
except ValueError:
   # ignore error with a pass keyword
   pass
```

See also:
The 'event' object

5.2.3.4 Global Script Modules

Your project can have a set of global script modules that any other script can reference. These modules all reside under the `app` namespace. These modules are available in bath Gateway and Client scope scripts. This is a great way to extract common logic into a central place - a core tenet of a well designed code.

To use your global script module, you'll need to have `app` imported into your current scope. Event handler scripts get this automatically, but user defined functions and other scripts will have to use `import app` to use global script modules.

See also:
Script Modules

5.2.3.5 Gateway vs Client Scripts

Your project can execute scripts under two different scopes: *Gateway* and *Client*.

Gateway scripts execute on the Ignition Gateway, which means that they will always execute in exactly one place. If you are using Clustering, they will only run on the cluster's *master node*.

Client scripts execute on *each* running Client. Because Clients are full-fledged applications, this means that the scripts are running on the computer running the client, not on the Gateway's host server computer. This means that they don't have access to the Gateway's filesystem, etc. It also means that if no clients are launched, the scripts will not be executing.

See also:
Project Event Scripts

5.2.3.6 Timer, Keystroke, and Tag Change Scripts

You can have scripts run for all sorts of global events. These are called project event scripts. You can have a script run every time a tag changes value, a key is pressed etc.

See also:
Project Event Scripts

5.2.3.7 Python Standard Library

You can use much of the Python standard library in Ignition. The available modules are listed here.

Learn more about the python standard library at http://www.python.org/doc/2.1/lib/lib.html

```
base64              htmlentitydefs      site
bdb                 htmllib             socket
bisect              httplib             sre
calendar            javaos              sre_compile
```

© 2011 Inductive Automation

```
cmd                 javapath            sre_constants
colorsys            linecache           sre_parse
commands            marshal             stat
ConfigParser        mimetypes           string
copy                ntpath              StringIO
copy_reg            nturl2path          tempfile
difflib             pdb                 urllib
dircache            pickle              urlparse
dospath             posixpath           UserDict
fileinput           pprint              UserList
fnmatch             Queue               UserString
formatter           random              xmllib
fpformat            re                  zipfile
ftplib              repr                zlib
gzip                shutil                future
```

5.2.3.8 Accessing Java

When programming Python in Ignition, your code executes in the Jython implementation of Python. (See About Scripting - Python or Jython?). While this doesn't have any great effect on the Python language itself, one of the great side benefits is that your Python code can seamlessly interact with Java code, *as if it were Python code*. This means that your Python code has access to the entire Java standard library, which is saying a lot.

To use Java classes, you simple import them as if they were Python modules. For example, the following code will print out all of the files in the user's home directory. This code uses the Java classes `java.lang.System` and `java.io.File` to look up the user's home directory and to list the files. Notice that we can even use the Python-style `for` loop to iterate over a Java sequence.

```
from java.lang import System
from java.io import File

homePath = System.getProperty("user.home")
homeDir = File(homePath)

for filename in homeDir.list():
   print filename
```

You can find the reference documentation for the Java standard class libraray (a.k.a. the "*JavaDocs*") here: http://java.sun.com/j2se/1.5.0/docs/api/index.html

Subclassing Java

You can even create Python classes that implement Java interfaces. If this is greek to you - don't worry, it isn't crucial. You'd need some understanding of Java and object-oriented programming concepts, which are outside the scope of this manual.

To create a Python class that implements a Java interface, you simply use the interface as a superclass for your Python class. For example, we could augment the example above to use the overload `java.io.File.list(FilenameFilter)`. To do this, we'll need to create a `FilenameFilter`, which is an interface in Java that defines a single function:

```
boolean accept(File dir, String name)
```

© 2011 Inductive Automation

To implement this interface, we create a Python class that has java.io.FilenameFilter as its superclass, and implements that Java-style function in a Python-esque way.

```
from java.lang import System
from java.io import *

class ExtensionFilter(FilenameFilter):
   def __init__(self, extension=".txt"):
      self.extension=extension.lower()

   def accept(self, directory, name):
      # make sure that the filename ends in the right extension
      return name.lower().endswith(self.extension)


homePath = System.getProperty("user.home")
homeDir = File(homePath)

# prints out all .txt files
for filename in homeDir.list(ExtensionFilter()):
   print filename

# prints out all .pdf files
for filename in homeDir.list(ExtensionFilter(".pdf")):
   print filename
```

5.3 Expressions

5.3.1 Overview

The expression language is used to define dynamic values for component properties and expression tags. Expressions often involve one or more other values that are used to calculate a final value. Expressions don't *do* anything, other than return a value.

The classic example for an expression is to change a temperature that is stored in Celsius to Fahrenheit in order to display it. Suppose you had a tag, `Tank6/Temp` that was in Celsius. If you wanted to display that tag in Fahrenheit on a Label, you would use an Expression Binding on the label's *text* property using the following expression:

```
1.8 * {Tank6/Temp} + 32
```

Every time that the temperature tag changes, the expression will re-calculate the value and push it into the Label's text property. Now lets say that you wanted to append a "°F" to the end of the label so that the user knew the units of the temperature. You could simply use some string concatenation in your expression, like this:

```
(1.8 * {Tank6/Temp} + 32) + " °F"
```

Lets suppose that you wanted to give the user an option to display the value in Celsius or Fahrenheit, based on checking a checkbox. You could add a Check Box component to th screen called DisplayFahrenheit. Then you could use this expression to dynamically display either unit, based upon the user's selection:

```
if({Root Container.DisplayFahrenheit.selected},
    (1.8 * {Tank6/Temp} + 32) + " °F",
```

© 2011 Inductive Automation

```
    {Tankf/Temp} +  " °C")
```

Where are Expressions Used?

Expressions are used in two major areas:

1. Expression Binding. The expression property binding is the most common area to use an expression. These expressions can reference tag values and property values in the same window.
2. Expression Tags. You can use an expression to dynamically calculate the value of a tag itself using a tag expression.
3. Expression Items. Expression items found in SQLBridge Transaction Groups. This uses the standard expression set with a few additions.

5.3.2 Syntax

As its name suggests, everything in the expression language is an "*expression*". This means that everything *returns a value*. `5` is an expression. So is `5+1`. So are `{MyTags/TankLevel}` and `{MyTags/TankLevel}+1`. Expressions can be combined in many powerful ways. Lets take a look at how expressions are written.

More formally, an expression is:

A Number
A Boolean
A String
A bound SQLTag
A bound property
A function call
A Dataset access
An equation involving any of these

Literal Values

Literal values are things like numbers, booleans, and strings that are represented directly in the language. In the expression language, numbers can by typed in directly as integers, floating point values, or using hexadecimal notation with a 0x prefix. Examples:

```
42
8.927
0xFFC2
```

Strings are represented by surrounding them with double or single quotes. You can use the backslash character to escape quotes that you want to be included in the string. Examples:

```
"This is a regular string"
'This one uses single quotes'
"This string uses \"escaping\" to include quotes inside the string"
```

Operators

You can use these arithmetic, logical, and bit-shifting operators to combine expressions.

-	Unary Minus	Negates a number.
!	Not	Logical opposite of a boolean
^	Power	Raises a number to the power of another number. `a^b` is a^b
%	Modulus	Modulus or remainder of two numbers. `a%b` is the *remainder* of `a÷b`
*	Multiply	
/	Divide	
+	Add /	If both operands are numbers, this will add them together. Otherwise treats

© 2011 Inductive Automation

	Concatenate	arguments as strings and performs concatenation.
-	Subtraction	
&	Bitwise AND	
|	Bitwise OR	
xor	Bitwise XOR	
<<	Left Shift	A signed bitwise left shift
>>	Right Shift	A signed bitwise right shift
>	Greater Than	Logical greater-than test between two numbers. Returns a boolean.
<	Less Than	
>=	Greater or Equal	
<=	Less or Equal	
=	Equal	Tests for equality between two operands.
!=	Not Equal	Tests for equality, returning true when not equal
&&	Logical AND	Returns true when both operands are true. Anything non-zero is considered true.
||	Logical OR	Returns true when either operand is true. Anything non-zero is considered true.
like	Fuzzy string matching	Compares the left-hand value with the pattern on the right side. The pattern may consist of %,*, and ? wildcards.

Bound Values

Bound values are paths to other values enclosed in braces. These will appear red in the expression editor. When you are writing an expression for a Expression Binding, you can reference tag values and property values using the brace notation. When you are writing an expression for an Expression Tag, you can only reference other tag values. You can use the Insert Property ( ) and Insert Tag Value ( ) buttons to build these references for you.

Dataset Access

If you have an expression that returns a Dataset, you can pull a value out of the datatset using the dataset access notation, which takes one of these forms:

Dataset_Expression ["Column_Name"] returns the value from the first row at the given column name

Dataset_Expression [Row_Index] returns the value from the given row at the first column

Dataset_Expression [Row_Index, "Column_Name"] returns the value from the given row at the given column name

Dataset_Expression [Row_Index, Column_Index] returns the value from the given row at the given column index

For example, this expression would pull a value out of a Table at row 6 for column "ProductCode":

```
{Root Container.Table.data}[6, "ProductCode"]
```

Note that you'll often have to convince the expression system that what you're doing is safe. The expression language can't tell what the datatype will be for a given column, so you may have to use a *type-casting* function to convince the expression language to accept your expression, like this:

```
toInt({Root Container.Table.data}[6, "ProductCode"])
```

Functions

The expression language's functions are where much of the real power lies. A function may take various arguments, all of which can themselves be any arbitrary expression. This means that you can use the results of one function as the argument to another function. In general, the syntax for a function call is:

functionName(expression1, expression2, ...)

The rest of this user manual section is devoted to the various functions available.

© 2011 Inductive Automation

Whitespace and Comments

Whitespace, such as spaces, tabs and newlines, are largely ignored in the expression language. It is often helpful to break your expression up onto multiple lines for clarity. Comments are delimited by two forward slashes. This will make the rest of that line be ignored. This example shows an `if` function spread over 4 lines with comments annotating the arguments.

```
if( {Root Container.UseTagValueOption.selected},
    {MyTags/SomeValue}, // Use the tag value
    "Not Selected",     // Use default value if the user doesn't check the box
)
```

© 2011 Inductive Automation

Deployment

Part VI

inductive automation
800.266.7798
www.inductiveautomation.com

6 Deployment

6.1 Licensing and Activation

One thing to consider when deploying an Ignition installation to production use is the manner in which it will be licensed.

If you anticipate that the installation might move from server to server frequently you may want to consider purchasing a USB license key to ease transition to new servers. This also makes things more convenient when the server is being deployed in an area without an active internet connection.

Otherwise a file-based licensing scheme can be used. If you have an internet connection you can activate the installation online. Otherwise you can download an activation request file and put it on a portable memory device and take it to a workstation with an active internet connection. From there you can upload the file to the Inductive Automation website and you will receive a license file, called `license.ipl`, in return. Take this file back to the gateway you are trying to activate and under `System > Licensing` you can upload and activate the license.

6.2 Making Backups

Backups can be made by going to `System > Backup/Restore` on the Ignition Gateway. Click the "Download Backup" button and save the file somewhere safe -- ideally somewhere that *DOES NOT* reside on the same machine running the gateway.

Backups save the user data inside the Ignition Gateway server. This includes all projects, drivers, images, and configuration, but not the modules.

6.3 Restoring from a Backup

Restoring from a backup is done under the `System > Backup/Restore` section on the Ignition Gateway. Click "Choose File", navigate to your backup file, and then click "Restore". The Gateway will need to restart itself to apply the restored settings.

See also:
Making Backups

6.4 Transferring Servers

There are only two things to consider when transferring your installation to a new server.

On The Old Server

1. Unactivate

If you are using a USB licensing key then this step is simple. Remove the USB key from this machine and transfer it to the other machine.

If not, you need to first visit the System > Licensing section of the Ignition Gateway and follow the link to unactivate. **This step is important.** If you do not unactive first you will either use up available activations, if your account has them, or need to contact support and get your installation unactivated manually over the phone.

2. Backup

© 2011 Inductive Automation

You need either a copy of the *most recent* backup (You are making backups, right? See the Making Backups section for more information) or to go ahead and make a backup at this point in time. This backup file is how you will transfer your existing settings to the new server.

On The New Server

1. Restore

Restore your settings using the backup file from the old server. Restoring from a backup is done under the `System > Backup/Restore` section on the Ignition Gateway. Click "Choose File", navigate to your backup file, and then click "Restore".

2. Activate

Activate your new installation of Ignition.

6.5 Gateway Homepage Customization

The Ignition Gateway home page can be customized to display only the information you want. On a new installation there are a number of panels that are helpful when beginning a project but when it comes time to deploy to production may no longer be necessary.

You can find these options on the "Homepage Config" tab in the `Configuration > Gateway Settings` section of the Ignition Gateway.

The following panels can be expanded/shrank or enabled/disabled:

Welcome to the Ignition Gateway
Launch Projects
Transaction Groups
Devices
Java Detection

6.6 Gateway Web Security

The Gateway's web interface can be secured in two ways: password protected sections and encrypted communication.

SSL

You can turn on SSL mode in the Gateway Configuration section under Configuration > Gateway Settings > Use SSL. This will make all communication for Clients, Designers, and web browsers using the web interface use encrypted SSL connections.

Password Protection

By default, the Configuration section is password protected, and this cannot be removed. You can also optionally protect the Status and the Home sections of the Gateway. You can also alter the *roles* that are required to access any of these sections. These settings are altered under Configuration > Gateway Settings.

6.7 Gateway Monitoring

The Ignition Gateway can be monitored in detail under the **Status section** or from the **Gateway Control Utility**.

© 2011 Inductive Automation

The Status section is broken down into the following parts.

Overview

The overview provides a top-down view of many of the components of your Gateway. This view is also useful for determining what step might be next when setting up your Ignition Gateway for the first time. You can view the status of your database connections, device connections, OPC connections, the number of open clients and the number of open designers.

Modules

A list of installed modules, their status, as well information about their version and current license state.

Clustering

Here you will find information about this Ignition Gateway pertaining to clustering, such as state, mode, number of messages and number of calls sent.

SQLTags

The SQLTags section lists information and statistics about all configured SQLTags Providers as well as a view into the SQLTags subscription model, scan classes, and what tags it is currently subscribed to.

Database Connections

This important section shows your database connections, their status, and their current load. Each status panel has information about the connection such as queries/second, total queries, and the average query duration.

Store & Forward

The Store & Forward section shows you what each of your S&F engines are doing. They show the number of pending and quarantined records that exist in the various stages of the S&F engines, as well as the throughput of records going through each stage. Note that if the final sink (the Database Storage) is available, data will jump directly from the memory buffer to the database storage, skipping the local cache.
See also: Data Quarantining.

OPC Connections

All of your OPC connections and any subscriptions you have made through these connections will be shown in this section. You can view the status of any connection as well as find details for trouble shooting when the status is bad. Statistics on the number of reads, writes, and updates are also kept.

Sessions

This section shows details about all of the Designer and Client sessions that are communicating with this Gateway. You can see detail about their subscriptions, user credentials, etc.

Ignition OPC-UA Server (Requires OPC-UA Module)

This section has two tabs of information.

The first tab is the Devices tab. Here you will find a list of all configured devices, the status for each device, as well as details about the driver that device is using.

The second tab is the Server Statistics tab. This is a set of basic performance statistics accumulated

© 2011 Inductive Automation

for the duration the server has been running as well as a list of subscriptions and their corresponding subscribed nodes that the server currently knows about.

6.8 Installing a Genuine SSL Certificate

When you turn on SSL in Ignition, the web browser uses what is called a "self-signed" certificate. This gives you the *encryption* benefits of SSL, but not the identity validation, and it isn't a 'real' certificate. This is why a web browser will display nasty warnings to users that they shouldn't trust the website.

We are not able to ship a real certificate with Ignition because SSL certificates have to be purchased individually from a **certificate authority**, such as Verisign, GoDaddy, or Comodo.

This guide will show you how to purchase and install a real SSL certificate from a certificate authority and install it in Ignition. You'll need to be comfortable executing command-line programs in order to complete this guide. The examples in this guide assume a Windows environment, but the general procedure would be identical in Linux.

1. Install the JDK

There are some command-line tools you'll need to use to create a certificate request and to install your certificate. These tools come with the Java Development Kit (JDK). It is likely that you only have the Java Runtime Environment (JRE) installed. Go to http://java.oracle.com and click on *Java SE*. Download the Java SE 6 JDK and install it.

2. Open a command prompt

Open a command prompt *(Start > Run > cmd)* and change directory into your JDK tools directory.

```
cd C:\Program Files\Java\jdk1.6.0_24\bin
```

3. Create your keystore

SSL certificates for Ignition are stored in a file called a *keystore*. You'll need to create your own keystore file with a certificate in it before you can purchase the SSL certificate.

a. Enter the following command:
```
keytool -genkey -alias tomcat -keyalg RSA -keysize 2048 -keystore C:\ssl.key
```
(you can put the file wherever you want for now but it should be called "ssl.key")

b. It will prompt you to enter a password. Use the password: `ignition`

c. You will then be prompted for your "first and last name". *Do not* actually use your first and last name. This value *must* be *one* of these for your Ignition Gateway:
   1. Fully Qualified Domain Name (e.g. "secure.yourdomain.com")
   2. Public IP address (e.g. "202.144.8.10")
   3. Full Server Name of your internal server (e.g. "scadaserver")
   4. Private IP address (e.g. "192.168.0.1")

d. It will then prompt you for information about your company. Input all data accurately, as the certificate authority will need to verify this information.

e. Lastly, it will ask you for the password for alias `<tomcat>`. Press `RETURN` to use the same password as the keystore file

4. Generate a Certificate Signing Request

At this point, you have a keystore file named "ssl.key" at the root of your C:\ drive (or wherever you

© 2011 Inductive Automation

specified it to be in step 3a)

In your command prompt window, enter this command:

```
keytool -certreq -alias tomcat -file C:\csr.txt -keystore C:\ssl.key
```

It will prompt you for the keystore password (`ignition`). You now have a certificate request file at `C:\csr.txt`

5. Buy the SSL certificate

Now you need to get your SSL certificate signed by a certificate authority. When you go to a certificate authority (Verisign, GoDaddy, Comodo, etc), they'll ask for your CSR, which is the `csr.txt` file that you created in step 4. Typically they'll ask you to paste your CSR into their web form. Open csr.txt in notepad, and copy-and-paste it into the certificate authority's form.

If prompted what software generated the CSR, choose Tomcat or Java

After the certificate authority has processed your payment and reviewed your CSR, they will send you your certificate via email.

6. Install the SSL certificate

After your SSL certificate has been emailed to you, you will want to follow the instructions provided for installing the certificate into a Java keystore. Your certificate authority will provide these instructions. The following is the procedure for installing a Comodo SSL certificate, provided as an example:

a. Extract the certificate files that were emailed to you, in this example they were extracted to C:\cert

b. Install the root certificate with the following command:
```
keytool -import -trustcacerts -alias root -file C:
\cert\AddTrustExternalCARoot.crt -keystore C:\ssl.key
```

c. Install the COMODO intermediate certificate:
```
keytool -import -trustcacerts -alias INTER -file C:\cert\COMODOHigh-
AssuranceSecureServerCA.crt -keystore C:\ssl.key
```

d. Install your server's certificate:
```
keytool -import -trustcacerts -alias tomcat -file C:\cert\192_168_1_7.
crt -keystore C:\ssl.key
```

7. Replace Ignition's default keystore

You now have a keystore file at `C:\ssl.key` that holds your SSL certificate. The certificate alias is "`tomcat`" and the password is "`ignition`". You can now replace the keystore file that ships with Ignition with your file. Make a backup of the file at `C:\Program Files\Inductive Automation\Ignition\tomcat\ssl.key` and replace it with your keystore file. You will need to restart the Ignition service after replacing this file.

Make sure your SSL port is allowed through your server's firewall. The default SSL port is 8043, and can be changed to the standard SSL port (443) through the Gateway Control Utilitiy (GCU).

If you have a redundant installation, you'll need to repeat this procedure on your backup server and buy a second certificate for it.

© 2011 Inductive Automation

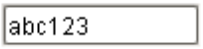
Appendix A. Components

Part VII

7 Appendix A. Components

7.1 Input

7.1.1 Text Field



A basic Text Field component

Description

The Text Field component is used for input of any single-line text. This component will accept any alpha-numeric input. If you're looking for a numeric field, see the Numeric Text Field.

This field features a *protected mode*. When you enable the `protectedMode` property, the field is not editable even when it recieves input focus. The user must *double click* on the field or *press enter* in order to edit the field. When they are done (press enter again or leave the field), the field becomes non-editable again.

The Text Field also supports the *reject updates during edit* feature. This feature ignores updates coming from property bindings while the component is being edited by a user.

Properties

Appearance

Font	Font of text of this component	
	Scripting name	`font`
	Data type	Font
Foreground Color	The foreground color of the component.	
	Scripting name	`foreground`
	Data type	Color
Background	The background color of the text box (when editable)	
	Scripting name	`editableBackground`
	Data type	Color
Non-Editable Background	The background color to use when this text box is non-editable	
	Scripting name	`nonEditableBackground`
	Data type	Color
	Flags	expert
Styles	Contains the component's styles	
	Scripting name	`styles`
	Data type	Dataset
	Flags	bindable | expert

Behavior

Editable?	If true, this is an input box, if false, this is display-only.	
	Scripting name	`editable`

© 2011 Inductive Automation

Data type	boolean

Defer Updates

When true, the 'text' property will not fire updates while typing, it will wait for Enter to be pressed.

Scripting name	`deferUpdates`
Data type	boolean

Protected Mode?

If true, users will need to double-click in the field in order to edit the text.

Scripting name	`protectedMode`
Data type	boolean

Reject Updates During Edit

If true, this field will not accept updates from external sources (like DB bindings)
while the user is editing the field.

Scripting name	`rejectUpdatesDuringEdit`
Data type	boolean
Flags	expert

Maximum Characters

The text box will be limited to this number of characters. Use -1 for unlimited.

Scripting name	`maxChars`
Data type	int

Touchscreen Mode

Controls when this input component responds if touchscreen mode is enabled.

Scripting name	`touchscreenMode`
Data type	int
Flags	expert
Values	`0` None
	`1` Single-Click
	`2` Double-Click

Common

Name

The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled

If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible

If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor

The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`

© 2011 Inductive Automation

Data type	int
Values	0 Default 1 Crosshair 2 Text 3 Wait 12 Hand 13 Move 4 SW Resize 5 SE Resize 6 NW Resize 7 NE Resize 8 N Resize 9 S Resize 10 W Resize 11 E Resize

Data

Text — Text of this component

Scripting name	`text`
Data type	String
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Layout

Horizontal Alignment — Determines the alignment of the label's contents along the X axis

Scripting name	`horizontalAlignment`
Data type	int
Flags	expert
Values	2 Left 0 Center 4 Right 10 Leading 11 Trailing

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- focus
- propertyChange
- key

Scripting Functions

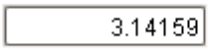
`getSelectedText()` Returns the currently selected or highlighted text in the text field.

Parameters	none
Returns	String

© 2011 Inductive Automation

7.1.2 Numeric Text Field

A basic Numeric Text Field



Numeric Text Field editing a floating-point value

Description

The Numeric Text Field is similar to the standard Text Field, except that it is specialized for use with numbers. Instead of a "text" property, it has four numeric "value" properties. Which one you use depends on the *mode* of the text box.

Like the standard Text Field, this text field can operate in *protected mode*. When you enable the protected property, the field is not editable even when it recieves input focus. The user must double click on the field or press enter in order to edit the field. When they are done (press enter again or leave the field), the field becomes non-editable again.

The Numeric Text Field also supports the *reject updates during edit* feature. This feature ignores updates coming from property bindings while the component is being edited by a user.

Properties

Appearance

Font	Font of text of this component	
	Scripting name	`font`
	Data type	Font
Foreground Color	The foreground color of the component.	
	Scripting name	`foreground`
	Data type	Color
Background	The background color of the text box (when editable)	
	Scripting name	`editableBackground`
	Data type	Color
Non-Editable Background	The background color to use when this text box is non-editable	
	Scripting name	`nonEditableBackground`
	Data type	Color
	Flags	expert
Decimal Format	The formatting string used for displaying numbers.	
	Scripting name	`decimalFormat`
	Data type	String
Styles	Contains the component's styles	
	Scripting name	`styles`
	Data type	Dataset
	Flags	bindable | expert

Behavior

Use Bounds?	Only allows user-entered values between a minimum and maximum. Unless you turn on "Error on out-of-bounds", user-entered values will be

© 2011 Inductive Automation

silently modified to be in-bounds.

Scripting name `useBounds`
Data type boolean

Error on Out-of-Bounds — Show an error message if the user input is out-of-bounds?

Scripting name `errorOnOutOfBounds`
Data type boolean

Out Of Bounds Message — The error message to display if input is out of bounds

Scripting name `outOfBoundsMessage`
Data type String
Flags expert

Editable? — If true, this is an input box, if false, this is display-only.

Scripting name `editable`
Data type boolean

Protected Mode? — If true, users will need to double-click in the field in order to edit the value.

Scripting name `protectedMode`
Data type boolean

Reject Updates During Edit — If true, this field will not accept updates from external sources (like DB bindings)
while the user is editing the field.

Scripting name `rejectUpdatesDuringEdit`
Data type boolean
Flags expert

Touchscreen Mode — Controls when this input component responds if touchscreen mode is enabled.

Scripting name `touchscreenMode`
Data type int
Flags expert
Values
- 0 None
- 1 Single-Click
- 2 Double-Click

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Enabled — If disabled, a component cannot be used.

Scripting name `enabled`
Data type boolean

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

© 2011 Inductive Automation

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor

The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default
	1 Crosshair
	2 Text
	3 Wait
	12 Hand
	13 Move
	4 SW Resize
	5 SE Resize
	6 NW Resize
	7 NE Resize
	8 N Resize
	9 S Resize
	10 W Resize
	11 E Resize

Data

Number Type

What type of numbers should this field accept?

Scripting name	`mode`
Data type	int
Values	**0** Integer
	3 Double
	1 Long
	2 Float

Maximum

The maximum value (inclusive), if useBounds is true.

Scripting name	`maximum`
Data type	double
Flags	bindable

Minimum

The minimum value (inclusive), if useBounds is true.

Scripting name	`minimum`
Data type	double
Flags	bindable

Value (Integer)

The value as an integer. Make sure you use the value property that corresponds to your Number Type setting.

Scripting name	`intValue`
Data type	int
Flags	bindable

Value (Double)

The value as a double. Make sure you use the value property that corresponds to your Number Type setting.

Scripting name	`doubleValue`
Data type	double
Flags	bindable

Value (Long)

The value as a long. Make sure you use the value property that corresponds to your Number Type setting.

Scripting name	`longValue`

© 2011 Inductive Automation

Data type	long
Flags	bindable

Value (Float) — The value as a float. Make sure you use the value property that corresponds to your Number Type setting.

Scripting name	`floatValue`
Data type	float
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Layout

Horizontal Alignment — Determines the alignment of the label's contents along the X axis

Scripting name	`horizontalAlignment`
Data type	int
Flags	expert
Values	**2** Left **0** Center **4** Right **10** Leading **11** Trailing

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- focus
- propertyChange
- key

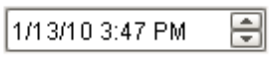
Scripting Functions

`getSelectedText()` Returns the currently selected or highlighted text in the text field.

Parameters	none
Returns	String

7.1.3 Spinner

A Spinner in Integer mode



A Spinner in Date mode

Description

The spinner component represents a value that is part of a series of values, such as numbers and dates. It allows you to not only edit the value directly, but to 'spin' the value up or down, using the up and down buttons that are part of the component. When setting up property bindings, make sure you use the value property that corresponds to the spinner mode. For example, if you chose the *Double* spinner mode, you should bind the `doubleValue` property.

© 2011 Inductive Automation

Properties

Appearance

Font

Font of text of this component

Scripting name: `font`
Data type: Font

Foreground Color

The foreground color of the component.

Scripting name: `foreground`
Data type: Color

Background Color

The background color of the component.

Scripting name: `background`
Data type: Color

Number Format

A number format pattern to use when the spinner is in numeric mode.

Scripting name: `numberFormat`
Data type: String

Date Format

A date format pattern to use when the spinner is in date mode.

Scripting name: `dateFormat`
Data type: String

Styles

Contains the component's styles

Scripting name: `styles`
Data type: Dataset
Flags: bindable | expert

Behavior

Spinner Mode

The mode controls which data type this spinner accepts

Scripting name: `spinnerMode`
Data type: int
Values:

0	Integer
1	Double
2	Date

Date Step Size

The amount to step up or down when in 'Date' mode.

Scripting name: `dateStepSize`
Data type: int
Values:

1	Year
2	Month
3	Week
5	Day
10	Hour
12	Minute
13	Second
14	Millisecond

Numeric Step Size

The size to step up or down when in 'Integer' or 'Double' mode.

Scripting name: `stepSize`
Data type: double

Touchscreen Mode

Controls when this input component responds if touchscreen mode is enabled.

Scripting name: `touchscreenMode`
Data type: int
Flags: expert

© 2011 Inductive Automation

Values	0	None
	1	Single-Click
	2	Double-Click

Common

Name — The name of this component.

Scripting name: `name`
Data type: String
Flags: bindable

Enabled — If disabled, a component cannot be used.

Scripting name: `enabled`
Data type: boolean

Visible — If disabled, the component will be hidden.

Scripting name: `visible`
Data type: boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name: `border`
Data type: Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name: `toolTipText`
Data type: String

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name: `opaque`
Data type: boolean

Data

Numeric Minimum — The minimum value this spinner will accept when in 'Integer' or 'Double' mode.

Scripting name: `minValue`
Data type: double

Numeric Maximum — The maximum value this spinner will accept when in 'Integer' or 'Double' mode.

Scripting name: `maxValue`
Data type: double

Value (Integer) — The current value if mode is 'Integer'

Scripting name: `intValue`
Data type: int
Flags: bindable

Value (Double) — The current value if mode is 'Double'

Scripting name: `doubleValue`
Data type: double
Flags: bindable

Value (Date) — The current value if mode is 'Date'

Scripting name: `dateValue`
Data type: Date
Flags: bindable

© 2011 Inductive Automation

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Uncategorized

Date in Milliseconds

Scripting name	`dateInMillis`
Data type	long
Flags	bindable | read-only

Scripting

Events

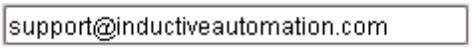
The following event sets are fired by this component. See Component Event Handlers to learn more.

propertyChange

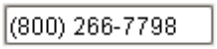
Scripting Functions

This component has no special scripting functions.

7.1.4 Formatted Text Field



With an email-address regular expression filter



With a US phone-number mask formatter

Description

This specialized text field is used for alphanumeric text input that must match some specific pattern or needs to be formatted in a specific way. It operates in two modes:

Formatted Mask

In this mode, input is automatically formatted and restricted based on a *format mask*. For example, a format mask like: `(###) ###-####` will allow the entry of a 10-digit US phone number. The formatting characters are automatically inserted if the user does not type them in. Any other characters are restricted. The following characters may be used in a formatted mask pattern:

- `#` Any valid number, Such as 0-9.
- `'` Escape character, used to escape any of the special formatting characters.
- `U` Any letter. All lowercase letters will be mapped to upper case automatically.
- `L` Any letter. All upper case letters will be mapped to lower case automatically.
- `A` Any letter or number.
- `?` Any letter, case is preserved.
- `*` Anything.
- `H` Any hex character (0-9, a-f or A-F).

Examples:

- `##U-####/UU` A product code with a specifc format, like `28E-8213/AR`
- `0xHHHH` A hex digit, automatically prepends "`0x`" no the front. e.g. "`0x82FF`"
- `#UUU###` A California license plate, eg. `4ABC123`

Regular Expression

In this mode, input is validated against a regular expression. A regular expression is a special string

© 2011 Inductive Automation

that defines a set of allowed strings. See http://en.wikipedia.org/wiki/Regular_expression. Any input that matches the given regular expression is allowed, and input that doesn't match, is restricted. And yes, while powerful, regular expressions are decidedly difficult to decipher.
Examples:

`\p{Upper}\p{Lower}*, \p{Upper}\p{Lower}*`	A name, formatted such as `Smith, John`
`\d{3}-\d{2}-\d{4}`	A US social security number, like `123-45-6789`
`\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}`	A network IPv4 address, like `67.82.120.116`
`^[a-f0-9A-F]{6}$`	A six-digit hexadecimal number.

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the component.

Scripting name	`background`
Data type	Color

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Validation Mode — Select regular expression or mask-driven field validation

Scripting name	`validationMode`
Data type	int
Values	**1** Regular Expression **2** Formatted Mask

Formatted Mask Pattern — Formatted Mask Validation Pattern

Scripting name	`formattedMaskPattern`
Data type	String

Reg Ex Pattern — Regular Expression Validation Pattern

Scripting name	`validationPattern`
Data type	String

Allows Invalid Text — Allows Invalid text to Commit

Scripting name	`allowsInvalid`
Data type	boolean

Overwrites Text — Overwrites text while typing

Scripting name	`overwriteMode`
Data type	boolean

Commit While Typing — Commits valid text while user is typing

Scripting name	`commitsOnValidEdit`

© 2011 Inductive Automation

Data type boolean

Focus Lost Behavior

Controls how a transaction can be committed

Scripting name `focusLostBehavior`
Data type int
Values
- 2 Revert
- 1 Commit or Revert
- 0 Commit
- 3 Persist

Touchscreen Mode

Controls when this input component responds if touchscreen mode is enabled.

Scripting name `touchscreenMode`
Data type int
Flags expert
Values
- 0 None
- 1 Single-Click
- 2 Double-Click

Common

Name

The name of this component.

Scripting name `name`
Data type String
Flags bindable

Enabled

If disabled, a component cannot be used.

Scripting name `enabled`
Data type boolean

Visible

If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Opaque

If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name `opaque`
Data type boolean

Cursor

The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values
- 0 Default
- 1 Crosshair
- 2 Text
- 3 Wait
- 12 Hand
- 13 Move
- 4 SW Resize
- 5 SE Resize
- 6 NW Resize

© 2011 Inductive Automation

7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Text — Contents of this Text Field

Scripting name	`text`
Data type	String

Committed Value — Committed Text Value

Scripting name	`committedValue`
Data type	String
Flags	bindable | expert

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Layout

Horizontal Alignment — Determines the alignment of the label's contents along the X axis

Scripting name	`horizontalAlignment`
Data type	int
Flags	expert
Values	2 Left
	0 Center
	4 Right
	10 Leading
	11 Trailing

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- focus
- propertyChange
- key

Scripting Functions

This component has no special scripting functions.

7.1.5 Password Field

A basic Password component

Description

A password field is like a text field that doesn't display the text that is being edited. You may alter the *echo character* (*) if you'd like.

© 2011 Inductive Automation

Properties

Appearance

Font
Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color
The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color
The background color of the component.

Scripting name	`background`
Data type	Color

Echo Character
The character that is displayed instead of the real ones.

Scripting name	`echoCharacter`
Data type	String

Styles
Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Touchscreen Mode
Controls when this input component responds if touchscreen mode is enabled.

Scripting name	`touchscreenMode`
Data type	int
Flags	expert
Values	**0** None **1** Single-Click **2** Double-Click

Common

Name
The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled
If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible
If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`

© 2011 Inductive Automation

Data type String

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name `opaque`
Data type boolean

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Text — Text of this component

Scripting name `text`
Data type String
Flags bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

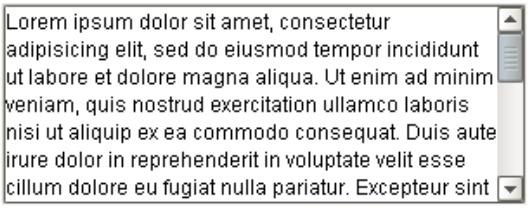
- mouse
- mouseMotion
- focus
- propertyChange
- key

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.1.6 Text Area


Lorem ipsum dolor sit amet, consectetur
adipisicing elit, sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute
irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint

A Text Area with line-wrap turned on

Description

Suitable for multi-line text display and editing. Will scroll vertically on demand. Will scroll horizontally if l*ine wrap* is off. Only supports plain-text, no HTML formatting or styled text.

Properties

Appearance

Font	Font of text of this component
Scripting name	`font`
Data type	Font
Foreground Color	The foreground color of the component.
Scripting name	`foreground`
Data type	Color
Background Color	The background color of the component.
Scripting name	`background`
Data type	Color
Rows	The number of rows you expect to display (used as a hint for scrollbars).
Scripting name	`rows`
Data type	int
Columns	The number of columns you expect to display (used as a hint for scrollbars).
Scripting name	`columns`
Data type	int
Styles	Contains the component's styles
Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Editable	Controls whether or not the user can edit the text within this text area.
Scripting name	`editable`
Data type	boolean
Defer Updates	If true, bindings will not affect the component's text while a user is editing the text.
Scripting name	`deferUpdates`
Data type	boolean

© 2011 Inductive Automation

Line Wrap

Should this area wrap lines?

Scripting name	`lineWrap`
Data type	boolean

Touchscreen Mode

Controls when this input component responds if touchscreen mode is enabled.

Scripting name	`touchscreenMode`
Data type	int
Flags	expert
Values	0 None 1 Single-Click 2 Double-Click

Common

Name

The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled

If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible

If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Opaque

If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name	`opaque`
Data type	boolean

Cursor

The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	0 Default 1 Crosshair 2 Text 3 Wait 12 Hand 13 Move 4 SW Resize 5 SE Resize 6 NW Resize 7 NE Resize 8 N Resize 9 S Resize 10 W Resize

© 2011 Inductive Automation

Data

Text	Text of this component
	Scripting name `text`
	Data type String
	Flags bindable
Data Quality	The data quality code for any tag bindings on this component.
	Scripting name `dataQuality`
	Data type int
	Flags bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

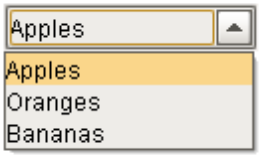
- mouse
- mouseMotion
- focus
- propertyChange
- key

Scripting Functions

This component has no special scripting functions.

7.1.7 Dropdown List

A Dropdown showing a selected value



A Dropdown showing its options

Description

The dropdown component is a great way to display a list of choices in a limited amount of space. The current selection is shown, and the choices are only presented when the user clicks on the dropdown button. The choices that are shown depend on the `data` property. This is a *dataset*, which can be typed in manually in the Designer, or (more commonly) it can be populated dynamically from a property binding, often a SQL Query binding.

It is often the case that you want to display choices to the user that are 'dressed up' versions of the actual choices. For instance, suppose that you are selecting choices for a downtime tracking entry. The choices might be: "Operator Error", "Machine Malfunction", and "Other". But, you really want to map these choices to some numeric code which is how the choice is stored. So, for instance, when the user chooses "Other" you really want to get the number 3. The dropdown component is perfect for such a use. The `data` property can be set up in one of three fashions, which control how the "selected values" properties change.

The 3 ways to set up the `data` dataset and the corresponding behavior is as follows:

© 2011 Inductive Automation

One Column
[String]

Apples
Oranges
Bananas

Dropdown displays values from the first column
Selected Value is undefined
Selected String Value represents value from first column
Selected Label represents value from first column

Two Columns
[Integer, String]

201	Apples
202	Oranges
203	Bananas

Dropdown displays values from second column
Selected Value represents value from first column
Selected String Value is undefined
Selected Label represents value from second column

Two Columns
[String, String]

APL	Apples
ORN	Oranges
BAN	Bananas

Dropdown displays values from second column
Selected Value is undefined
Selected String Value represents value from first column
Selected Label represents value from second column

The dropdown component can operate in one of three *Selection Modes*. These modes affect how the dropdown's current selection (defined by the values of its Selected Value, Selected String Value, and Selected Label properties) behave when the selection properties are set to values not present in the choice list, or conversely, when the choice list is set to a new dataset that doesn't contain the current selection:

Strict. Selected values must always correlate to an option in the list defined by the Data property. If an invalid selection is set (via a binding or a script), the selection will be set to the values defined by the No Selection properties. If the Data property is set to a list that does not contain the current selection, the current selection will be reset to the No Selection values.

Lenient. *(default)* Selected values are independent of the list defined by the Data property. This mode is useful to avoid race conditions that can cause problems in Strict mode when both the Data and the Selected Value properties are bound. If the current selection is not present in the Data list, the read-only property Selected Index will be -1.

Editable. The same selection rules as defined by Lenient mode, except that the dropdown itself becomes editable, allowing a user to type in their own arbitrary value. This value will be set as the dropdown's Selected Label.

Properties

Appearance

Font — Font of text of this component
Scripting name: `font`
Data type: Font

Foreground Color — The foreground color of the component.
Scripting name: `foreground`
Data type: Color

Background Color — The background color of the component.
Scripting name: `background`
Data type: Color

Selection Background
The background color of a selected cell in the dropdown list.

Scripting name	`selectionBackground`
Data type	Color
Flags	expert

Dropdown Display Mode
Changes the dropdown's display

Scripting name	`mode`
Data type	int
Values	0 List 1 Table

Max Row Count
The number of rows to display in the dropdown list before displaying a scrollbar.

Scripting name	`maximumRowCount`
Data type	int
Flags	expert

Hide Table Columns?
A comma separated list of columns to hide from the dropdown table, eg. 0,2 (only used in table mode)

Scripting name	`hideTableColumns`
Data type	String
Flags	expert

Show Table Header?
Selects whether or not the dropdown table header is displayed. (only used in table mode)

Scripting name	`showTableHeader`
Data type	boolean
Flags	expert

Max Table Width
The maximum width allowed for the dropdown table. (only used in table mode)

Scripting name	`maxTableWidth`
Data type	int
Flags	expert

Max Table Height
The maximum height allowed for the dropdown table. (only used in table mode)

Scripting name	`maxTableHeight`
Data type	int
Flags	expert

Styles
Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Selection Mode
The selection mode determines the behavior of the dropdown: whether its selected value must strictly be in the underlying set of choices, whether it is flexible, or even user-editable.

Scripting name	`selectionMode`
Data type	int
Values	0 Strict 1 Lenient 2 Editable

No Selection Value
The value when nothing is selected.

© 2011 Inductive Automation

Scripting name noSelectionValue
Data type int
Flags expert

No Selection String — The string value when nothing is selected.

Scripting name noSelectionString
Data type String
Flags expert

No Selection Label — The label to display when nothing is selected.

Scripting name noSelectionLabel
Data type String
Flags expert

Common

Name — The name of this component.

Scripting name name
Data type String
Flags bindable

Enabled — If disabled, a component cannot be used.

Scripting name enabled
Data type boolean

Visible — If disabled, the component will be hidden.

Scripting name visible
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name border
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name toolTipText
Data type String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name cursorCode
Data type int
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Data — The data which fills up the combo box. Either a 1 or 2 column DataSet,

© 2011 Inductive Automation

with the first column being the value, and the second being the display

Scripting name `data`
Data type Dataset
Flags bindable

Selected Value — The currently selected value

Scripting name `selectedValue`
Data type Integer
Flags bindable

Selected String Value — The currently selected value, if the value column is a string

Scripting name `selectedStringValue`
Data type String
Flags bindable

Selected Label — The currently selected label

Scripting name `selectedLabel`
Data type String
Flags bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Layout

Horizontal Alignment — Determines the alignment of the contents along the X axis

Scripting name `horizontalAlignment`
Data type int
Flags expert
Values
- `2` Left
- `0` Center
- `4` Right
- `10` Leading
- `11` Trailing

Vertical Alignment — Determines the alignment of the contents along the Y axis

Scripting name `verticalAlignment`
Data type int
Flags expert
Values
- `1` Top
- `0` Center
- `3` Bottom

Uncategorized

Selected Index — The index of the selected item.

Scripting name `selectedIndex`
Data type int
Flags bindable | read-only

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion

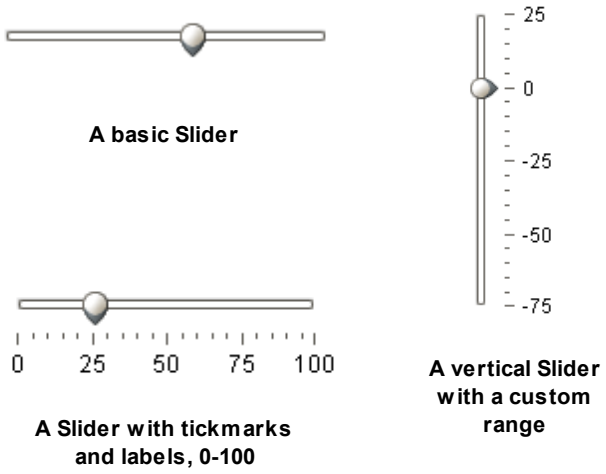
© 2011 Inductive Automation

focus
propertyChange
key

Scripting Functions

This component has no special scripting functions.

7.1.8 Slider



A basic Slider

A Slider with tickmarks and labels, 0-100

A vertical Slider with a custom range

Description

The slider component lets the user drag an indicator along a scale to choose a value in a range. The **Defer Updates** property determines whether or not the slider's **Value** changes as the user drags the mouse, or whether it waits until the user drops the slider handle.

Properties

Appearance

Font	Font of text of this component
	Scripting name `font`
	Data type Font
Foreground Color	The foreground color of the component.
	Scripting name `foreground`
	Data type Color
Background Color	The background color of the component.
	Scripting name `background`
	Data type Color
	Flags expert
Horizontal Slider	If true, slider is horizontal. Otherwise, it's vertical
	Scripting name `horizontal`
	Data type boolean
Major Tick Spacing	The distance, measured in values, between each major tick mark
	Scripting name `majorTickSpacing`
	Data type int

© 2011 Inductive Automation

Minor Tick Spacing
The distance, measured in values, between each minor tick mark
Scripting name minorTickSpacing
Data type int

Paint Track?
If true, the trac of the slider will be shown.
Scripting name paintTrack
Data type boolean

Paint Labels?
If true, value labels will be shown.
Scripting name paintLabels
Data type boolean

Paint Ticks?
If true, value tick marks will be shown.
Scripting name paintTicks
Data type boolean

Styles
Contains the component's styles
Scripting name styles
Data type Dataset
Flags bindable | expert

Behavior

Defer Updates
Only publish updates to value when not actively being changed.
Scripting name deferred
Data type boolean

Snap To Ticks?
Scripting name snapToTicks
Data type boolean

Inverted?
Specify true to reverse the value range shown for the slider and false to put the value range in the normal order.
Scripting name inverted
Data type boolean

Common

Name
The name of this component.
Scripting name name
Data type String
Flags bindable

Enabled
If disabled, a component cannot be used.
Scripting name enabled
Data type boolean

Visible
If disabled, the component will be hidden.
Scripting name visible
Data type boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name border
Data type Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.

© 2011 Inductive Automation

Scripting name `toolTipText`
Data type String

Opaque — If true, the background of the slider is drawn.

Scripting name `opaque`
Data type boolean
Flags expert

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Value — The current value of the slider.

Scripting name `value`
Data type int
Flags bindable

Minimum Value — The value when the slider is all the way left or down

Scripting name `minimum`
Data type int
Flags bindable

Maximum Value — The value when the slider is all the way right or up

Scripting name `maximum`
Data type int
Flags bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- focus
- propertyChange
- key

Scripting Functions

This component has no special scripting functions.

7.2 Buttons

7.2.1 Button

A standard push button

Buttons can have images

Buttons can be only images

Buttons can display state

Description

The Button component is a versatile component, often used for things like opening/closing windows, writing to tags, and triggering any sort of scripting logic. It can be used for showing status, as well. For example, if you have three buttons, Hand, Off, and Auto, not only can they set those modes, but their background color can display the current mode, although you'd be better off using the Multi-State Button for this.

To get buttons to do things, you add an event handler to the `actionPerformed` event. Many new users to the 1.0.0 module will configure an event handler for the `mouseClicked` event instead. While this will work, it is better to use the `actionPerformed` event. Why? Buttons can also be activated by tabbing over to them and hitting the space key, or they could be activated by pressing Alt and the button's *mnemonic* character. So, to make sure that your button works in all of these cases, configure your event handler on the `actionPerformed` event, not the `mouseClicked` event.

See also:
Typical Navigation Strategy
Event Types

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the button.

Scripting name	`buttonBG`
Data type	Color

Background 3D? — Should this button have a 3d type background, or a flat color one?

Scripting name	`background3D`
Data type	boolean
Flags	expert

Fill Area? — Controls whether or not this button's internal area is filled

Scripting name	`contentAreaFilled`
Data type	boolean
Flags	expert

© 2011 Inductive Automation

Border Painted? — Should the border of this button be displayed?

Scripting name	`borderPainted`
Data type	boolean
Flags	expert

Text — Text of this component

Scripting name	`text`
Data type	String
Flags	bindable

Image Path — The relative path of the image.

Scripting name	`path`
Data type	String
Flags	bindable

Disabled Image Path — The relative path of the image to be displayed when this component is not enabled.

Scripting name	`disabledPath`
Data type	String
Flags	expert

Icon-Text Spacing — The space (in pixels) between the icon (if any) and the text (if any)

Scripting name	`iconTextGap`
Data type	int

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Rollover — If true, the button may indicate that the mouse is hovering over it.

Scripting name	`rolloverEnabled`
Data type	boolean
Flags	expert

Focusable — If a button is not focusable, you will not be able to interact with it with the keyboard. This means you can't "tab" over to it.

Scripting name	`focusable`
Data type	boolean
Flags	expert

Mnemonic — A single letter that will activate the button using 'ALT-mnemonic'.

Scripting name	`mnemonicChar`
Data type	String

Default Button — If true, this button will be activated when the user presses enter on the window.

Scripting name	`defaultBtn`
Data type	boolean
Flags	expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

© 2011 Inductive Automation

Enabled
If disabled, a component cannot be used.
Scripting name `enabled`
Data type boolean

Visible
If disabled, the component will be hidden.
Scripting name `visible`
Data type boolean

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name `toolTipText`
Data type String

Cursor
The mouse cursor to use when hovering over this component.
Scripting name `cursorCode`
Data type int
Values
`0` Default
`1` Crosshair
`2` Text
`3` Wait
`12` Hand
`13` Move
`4` SW Resize
`5` SE Resize
`6` NW Resize
`7` NE Resize
`8` N Resize
`9` S Resize
`10` W Resize
`11` E Resize

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name `border`
Data type Border
Flags expert

Opaque
Is this button completely opaque? Most aren't, so this should usually be false.
Scripting name `opaque`
Data type boolean
Flags expert

Data

Data Quality
The data quality code for any tag bindings on this component.
Scripting name `dataQuality`
Data type int
Flags bindable | expert

Layout

Margin
The space between a button's text and its borders.
Scripting name `margin`
Data type Insets
Flags expert

Horizontal Alignment
The horizontal alignment of the button's contents (text and/or image)
Scripting name `horizontalAlignment`

© 2011 Inductive Automation

Data type int
Values
2 Left
0 Center
4 Right
10 Leading
11 Trailing

Horizontal Text Position — The horizontal position of the button's text relative to its image

Scripting name horizontalTextPosition
Data type int
Flags expert
Values
2 Left
0 Center
4 Right
10 Leading
11 Trailing

Vertical Alignment — The vertical alignment of the button's contents (text and/or image)

Scripting name verticalAlignment
Data type int
Values
1 Top
0 Center
3 Bottom

Vertical Text Position — The vertical position of the button's text relative to its image

Scripting name verticalTextPosition
Data type int
Flags expert
Values
1 Top
0 Center
3 Bottom

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
action
focus
propertyChange
key

Scripting Functions

doClick() — Virtually "clicks" the button, meaning that its actionPerformed event handler will run.

Parameters none
Returns nothing

7.2.2 2 State Toggle

The default "on" style

The default "off" style

Styles are highly customizable

© 2011 Inductive Automation

Description

This button is similar to the basic Toggle Button, but more finely tuned to work in realistic controls environments. Use this button any time you want to toggle a value between two states, such as On/Off, Stop/Run, etc. If you have more than two states (for example, Hand/Off/Auto, use the Multi-State Button).

If you have a tag whose value you want to toggle between 2 values (like zero and one), you can simply drag and drop the tag onto the button. This will bind both the *Control Value* and *Indicator Value* properties to that tag. Now set the *State 1 Value* and *State 2 Value* to your two states (they default to zero and one, respectively). Lastly, use the Styles Customizer to define the styles for your two states.

This button has four integer values that you use to set it up: the *Control Value*, the *Indicator Value*, and values that define the 2 different states: *State 1 Value* and *State 2 Value*. Every time you press the button, one of the state values is written to the control value. The Indicator Value is used to determine which state you're in. For example, suppose that State 1 Value was zero and State 2 Value is one. If Indicator Value is zero and you press the button, it'll write a one to the Control Value. The Style of the component is typically driven by the read-only property *Current State*. Current State equals zero when Indicator Value=State 1 Value and one otherwise.

See also:
Bidirectional Bindings
Component Styles

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the button.

Scripting name	`buttonBG`
Data type	Color

Background 3D? — Should this button have a 3d type background, or a flat color one?

Scripting name	`background3D`
Data type	boolean
Flags	expert

Fill Area? — Controls whether or not this button's internal area is filled

Scripting name	`contentAreaFilled`
Data type	boolean
Flags	expert

Border Painted? — Should the border of this button be displayed?

Scripting name	`borderPainted`
Data type	boolean
Flags	expert

© 2011 Inductive Automation

Text — Text of this component

Scripting name	`text`
Data type	String
Flags	bindable

Image Path — The relative path of the image.

Scripting name	`path`
Data type	String
Flags	bindable

Disabled Image Path — The relative path of the image to be displayed when this component is not enabled.

Scripting name	`disabledPath`
Data type	String
Flags	expert

Icon-Text Spacing — The space (in pixels) between the icon (if any) and the text (if any)

Scripting name	`iconTextGap`
Data type	int

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Rollover — If true, the button may indicate that the mouse is hovering over it.

Scripting name	`rolloverEnabled`
Data type	boolean
Flags	expert

Focusable — If a button is not focusable, you will not be able to interact with it with the keyboard. This means you can't "tab" over to it.

Scripting name	`focusable`
Data type	boolean
Flags	expert

Confirm? — If true, a confirmation box will be shown.

Scripting name	`confirm`
Data type	boolean

Confirm Text — The message to ask the user if confirmation is turned on.

Scripting name	`confirmText`
Data type	String

Mnemonic — A single letter that will activate the button using 'ALT-mnemonic'.

Scripting name	`mnemonicChar`
Data type	String
Flags	expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

© 2011 Inductive Automation

Scripting name `enabled`
Data type boolean

Visible

If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor

The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border
Flags expert

Opaque

Is this button completely opaque? Most aren't, so this should usually be false.

Scripting name `opaque`
Data type boolean
Flags expert

Data

Data Quality

The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Control Value

Bind this to the tag that controls the state. (Typically, this is bound to the same location as Indicator Value)

Scripting name `controlValue`
Data type int
Flags bindable

Indicator Value

Bind this to the tag that indicates the current state. (Typically, this is bound to the same location as Control Value)

Scripting name `indicatorValue`
Data type int

© 2011 Inductive Automation

Flags bindable

State 1 Value

The value that will be written to controlValue when the button is pushed in state 2.

Scripting name `state1Value`
Data type int
Flags bindable

State 2 Value

The value that will be written to controlValue when the button is pushed in state 1.

Scripting name `state2Value`
Data type int
Flags bindable

Current State

Read-only property that shows what state (0 or 1) this button is currently in.

Scripting name `state`
Data type int
Flags bindable | expert

Layout

Margin

The space between a button's text and its borders.

Scripting name `margin`
Data type Insets
Flags expert

Horizontal Alignment

The horizontal alignment of the button's contents (text and/or image)

Scripting name `horizontalAlignment`
Data type int
Values
- `2` Left
- `0` Center
- `4` Right
- `10` Leading
- `11` Trailing

Horizontal Text Position

The horizontal position of the button's text relative to its image

Scripting name `horizontalTextPosition`
Data type int
Flags expert
Values
- `2` Left
- `0` Center
- `4` Right
- `10` Leading
- `11` Trailing

Vertical Alignment

The vertical alignment of the button's contents (text and/or image)

Scripting name `verticalAlignment`
Data type int
Values
- `1` Top
- `0` Center
- `3` Bottom

Vertical Text Position

The vertical position of the button's text relative to its image

Scripting name `verticalTextPosition`
Data type int
Flags expert
Values
- `1` Top
- `0` Center
- `3` Bottom

© 2011 Inductive Automation

Scripting

Events

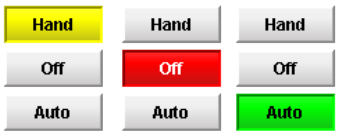
The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- action
- focus
- propertyChange
- key

Scripting Functions

This component has no special scripting functions.

7.2.3 Multi-State Button



3 Multi-State Buttons showing the default stettings

Many configurations are possible

Description

This button is really a series of two or more buttons, arranged in a column, row, or grid. Each button represents an integer-valued state. Each state defines two styles for a button: the selected style, and the unselected style. Each button is automatically displayed with the correct style based on the current state (the value of Indicator Value). When a button is pressed, it's state's value is written to the Control Value.

To configure a Multi-State Button, simply drag a tag that represents your state onto the Multi-State Button. This will bind both the Control Value and Indicator Value to that tag. Now open up the Multi-State Button customizer, and define your states: their order, values and styles. Lastly choose if you want the buttons to be a column, row, or grid by setting the Display Style property.

See also:
Bidirectional Bindings
Component Customizers

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Display Style — The display style (rows or columns) for this N-state button.

Scripting name	`displayStyle`
Data type	int
Values	0 Column

© 2011 Inductive Automation

	1 Row 2 Grid
Horizontal Gap	The horizontal spacing between buttons Scripting name `hGap` Data type int
Vertical Gap	The vertical spacing between buttons Scripting name `vGap` Data type int
Grid Rows	The number of rows if the Display Style is set to "Grid" mode. Scripting name `gridRows` Data type int
Grid Cols	The number of columns if the Display Style is set to "Grid" mode. Scripting name `gridCols` Data type int
Background 3D?	Controls whether or not the buttons have a gradient-style background color. Scripting name `background3D` Data type boolean Flags expert

Behavior

Confirm?	If true, a confirmation box will be shown. Scripting name `confirm` Data type boolean
Confirm Text	The message to ask the user if confirmation is turned on. Scripting name `confirmText` Data type String
States	A Dataset that stores the information for the different states. Scripting name `states` Data type Dataset Flags expert
Rollover	If true, the button may indicate that the mouse is hovering over it. Scripting name `rolloverEnabled` Data type boolean Flags expert
Focusable	If a button is not focusable, you will not be able to interact with it with the keyboard. This means you can't "tab" over to it. Scripting name `focusableEnabled` Data type boolean Flags expert

Common

Name	The name of this component. Scripting name `name` Data type String Flags bindable
Enabled	If disabled, a component cannot be used. Scripting name `enabled`

© 2011 Inductive Automation

Data type boolean

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Control Value — Bind this to the tag that controls the state. (Typically, this is bound to the same location as Indicator Value)

Scripting name `controlValue`
Data type int
Flags bindable

Indicator Value — Bind this to the tag that indicates the current state. (Typically, this is bound to the same location as Control Value)

Scripting name `indicatorValue`
Data type int
Flags bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- propertyChange
- key

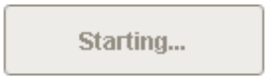
Scripting Functions

© 2011 Inductive Automation

This component has no special scripting functions.

7.2.4 One-Shot Button

A One-Shot button, waiting to be pressed



A One-Shot button, waiting for a PLC reset

Description

The latched button is great for telling a PLC to do something. It simply writes a value, and then waits for it to be reset by the PLC before it is available again. Note that this is only applicable when the PLC is programmed to reset the value after reading it. If your PLC expects the HMI to reset the bit, use the Momentary Button. Also note that this component is considered safer than the momentary button, because it receives positive feedback from the PLC that the signal was received, avoiding the timing dangers associated with a Momentary Button.

To use the latched button, bind an OPC tag bidirectionally to the latched button's Value property. When clicked, the button will write the value in its Set Value property to the Value property. Typically, Set Value is 1, and Value is 0 in a ready state, although the logic could be reversed or change simply by altering Set Value. The button can disable itself when it is writing, and will display different text. Note that the button considers itself to be writing whenever Value equals Set Value - you must make sure that the PLC resets this value, otherwise the button will remain in a writing state.

See also:
Bidirectional Bindings

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the button.

Scripting name	`buttonBG`
Data type	Color

Background 3D? — Should this button have a 3d type background, or a flat color one?

Scripting name	`background3D`
Data type	boolean
Flags	expert

Fill Area? — Controls whether or not this button's internal area is filled

Scripting name	`contentAreaFilled`
Data type	boolean
Flags	expert

© 2011 Inductive Automation

Border Painted?
Should the border of this button be displayed?
Scripting name `borderPainted`
Data type boolean
Flags expert

Image Path
The relative path of the image.
Scripting name `path`
Data type String
Flags bindable

Disabled Image Path
The relative path of the image to be displayed when this component is not enabled.
Scripting name `disabledPath`
Data type String
Flags expert

Icon-Text Spacing
The space (in pixels) between the icon (if any) and the text (if any)
Scripting name `iconTextGap`
Data type int

Styles
Contains the component's styles
Scripting name `styles`
Data type Dataset
Flags bindable | expert

Behavior

Rollover
If true, the button may indicate that the mouse is hovering over it.
Scripting name `rolloverEnabled`
Data type boolean
Flags expert

Focusable
If a button is not focusable, you will not be able to interact with it with the keyboard. This means you can't "tab" over to it.
Scripting name `focusable`
Data type boolean
Flags expert

Idle Text
The text of the button while its value is not being written
Scripting name `normalText`
Data type String
Flags bindable

Writing Text
The text of the button while its vaule is being written
Scripting name `writePendingText`
Data type String
Flags bindable

Disable While Writing
If true, the button will be disabled while it is writing.
Scripting name `disableWhileWriting`
Data type boolean

Confirm?
If true, a confirmation box will be shown.
Scripting name `confirm`
Data type boolean

Confirm Text
The message to ask the user if confirmation is turned on.
Scripting name `confirmText`

© 2011 Inductive Automation

Data type String

Mnemonic

A single letter that will activate the button using 'ALT-mnemonic'.

Scripting name `mnemonicChar`
Data type String
Flags expert

Common

Name

The name of this component.

Scripting name `name`
Data type String
Flags bindable

Enabled

If disabled, a component cannot be used.

Scripting name `enabled`
Data type boolean

Visible

If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor

The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border
Flags expert

Opaque

Is this button completely opaque? Most aren't, so this should usually be false.

Scripting name `opaque`
Data type boolean
Flags expert

Data

Data Quality

The data quality code for any tag bindings on this component.

© 2011 Inductive Automation

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Value — The current value. Should be bound bi-directionally to a tag

Scripting name	`value`
Data type	int
Flags	bindable

Set Value — The value to set the control value to when the button is pushed.

Scripting name	`setValue`
Data type	int
Flags	bindable

Layout

Margin — The space between a button's text and its borders.

Scripting name	`margin`
Data type	Insets
Flags	expert

Horizontal Alignment — The horizontal alignment of the button's contents (text and/or image)

Scripting name	`horizontalAlignment`
Data type	int
Values	**2** Left **0** Center **4** Right **10** Leading **11** Trailing

Horizontal Text Position — The horizontal position of the button's text relative to its image

Scripting name	`horizontalTextPosition`
Data type	int
Flags	expert
Values	**2** Left **0** Center **4** Right **10** Leading **11** Trailing

Vertical Alignment — The vertical alignment of the button's contents (text and/or image)

Scripting name	`verticalAlignment`
Data type	int
Values	**1** Top **0** Center **3** Bottom

Vertical Text Position — The vertical position of the button's text relative to its image

Scripting name	`verticalTextPosition`
Data type	int
Flags	expert
Values	**1** Top **0** Center **3** Bottom

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse

© 2011 Inductive Automation

mouseMotion
action
focus
propertyChange
key

Scripting Functions

This component has no special scripting functions.

7.2.5 Momentary Button

Momentary Button waiting to be pressed

Activated Momentary Button

Description

Momentary buttons are used to set a value for either a fixed amount of time, or however long the button remains held down, whichever is longer. Once the button is released, or the minimum time expires, the value is reset.

The momentary button uses it's *Control Value* property to affect the underlying data. Typically, this property uses a bidirectional tag binding to an OPC tag. When pressed, it will write its *On Value* to *Control Value*. When released, it will either write *Off Value* to *Control Value* immediately, or wait until *On Time* has elapsed (since the pressed event).

The button's *Indicator Value*, which is typically bound to the same OPC tag as *Control Value*, is used to draw an "active" indication border around the button. This gives the operator positive feedback that the value has written successfully. It also lets an operator at one terminal know if an operator at a different terminal is using the button currently.

⚠ Note that you may want to use the Latched Button instead of the Momentary Button if you simply need to send a signal to a PLC, and the PLC is able to reset the value. This lets the PLC reset the value, avoiding the potential for the bit to be left high. This is possible with the Momentary Button if, for example, the power to the client was cut while the button was held down.

Properties

Appearance

Font	Font of text of this component	
	Scripting name	`font`
	Data type	Font
Foreground Color	The foreground color of the component.	
	Scripting name	`foreground`
	Data type	Color
Background Color		
	Scripting name	`buttonBG`
	Data type	Color
Background 3D?	Should this button have a 3d type background, or a flat color one?	
	Scripting name	`background3D`

© 2011 Inductive Automation

Data type	boolean
Flags	expert

Fill Area?

Controls whether or not this button's internal area is filled

Scripting name	`contentAreaFilled`
Data type	boolean
Flags	expert

Rollover?

If true, the button may indicate that the mouse is hovering over it.

Scripting name	`rolloverEnabled`
Data type	boolean
Flags	expert

Text

Text of this component

Scripting name	`text`
Data type	String
Flags	bindable

Indicator Width

The width of the indication border that shows whether or not the indicator value is currently set.

Scripting name	`indicatorWidth`
Data type	int

On Color

The color of the indicator border when the indicator value is on.

Scripting name	`onColor`
Data type	Color

Off Color

The color of the indicator border when the indicator value is off

Scripting name	`offColor`
Data type	Color

Image Path

The relative path of the image.

Scripting name	`path`
Data type	String
Flags	bindable

Disabled Image Path

The relative path of the image to be displayed when this component is not enabled.

Scripting name	`disabledPath`
Data type	String
Flags	expert

Icon-Text Spacing

The space (in pixels) between the icon (if any) and the text (if any)

Scripting name	`iconTextGap`
Data type	int

Styles

Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Mnemonic

A single letter that will activate the button using 'ALT-mnemonic'.

Scripting name	`mnemonicChar`
Data type	String

On Value

The value that will be written to the Control Value on mouse-down.

Scripting name	`onValue`

© 2011 Inductive Automation

Data type int

Off Value
The value that will be written to the Control Value on mouse-up
Scripting name offValue
Data type int

On Time
The minimum amount of time to keep the control value at the "On Value"
Scripting name onTime
Data type int

Common

Name
The name of this component.
Scripting name name
Data type String
Flags bindable

Enabled
If disabled, a component cannot be used.
Scripting name enabled
Data type boolean

Visible
If disabled, the component will be hidden.
Scripting name visible
Data type boolean

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name toolTipText
Data type String

Border
The border surrounding this component.
Scripting name innerBorder
Data type Border

Cursor
The mouse cursor to use when hovering over this component.
Scripting name cursorCode
Data type int
Values
0 Default
1 Crosshair
2 Text
3 Wait
12 Hand
13 Move
4 SW Resize
5 SE Resize
6 NW Resize
7 NE Resize
8 N Resize
9 S Resize
10 W Resize
11 E Resize

Data

Control Value
Bind this to the tag that you want to control. (Typically, this is bound to the same location as Indicator Value)
Scripting name controlValue
Data type int
Flags bindable

© 2011 Inductive Automation

Indicator Value — Bind this to the tag that indicates the current state of the control value. (Typically, this is bound to the same location as Control Value)

Scripting name	`indicatorValue`
Data type	int
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Layout

Margin — The space between a button's text and its borders.

Scripting name	`margin`
Data type	Insets
Flags	expert

Horizontal Alignment — The horizontal alignment of the button's contents (text and/or image)

Scripting name	`horizontalAlignment`
Data type	int
Values	**2** Left **0** Center **4** Right **10** Leading **11** Trailing

Horizontal Text Position — The horizontal position of the button's text relative to its image

Scripting name	`horizontalTextPosition`
Data type	int
Flags	expert
Values	**2** Left **0** Center **4** Right **10** Leading **11** Trailing

Vertical Alignment — The vertical alignment of the button's contents (text and/or image)

Scripting name	`verticalAlignment`
Data type	int
Values	**1** Top **0** Center **3** Bottom

Vertical Text Position — The vertical position of the button's text relative to its image

Scripting name	`verticalTextPosition`
Data type	int
Flags	expert
Values	**1** Top **0** Center **3** Bottom

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion

© 2011 Inductive Automation

action
focus
propertyChange
key

Scripting Functions

This component has no special scripting functions.

7.2.6 Toggle Button

The two states of a toggle button

Description

The toggle button represents a bit: on (selected) or off (not selected). Visually the button looks down or depressed when it is selected, and up when it is not selected. Logically, this component is very similar to the Check Box component. Note that for implementing a controls screen, the 2 State Toggle is usually more appropriate than this component.

Properties

Appearance

Font	Font of text of this component Scripting name `font` Data type Font
Foreground Color	The foreground color of the component. Scripting name `foreground` Data type Color
Background Color	The background color for the button. Scripting name `buttonBG` Data type Color
Background 3D?	Should this button have a 3d type background, or a flat color one? Scripting name `background3D` Data type boolean Flags expert
Opaque	Set this to false if you want the button to be completely opaque. Scripting name `opaque` Data type boolean Flags expert
Fill Area?	Controls whether or not this button's internal area is filled Scripting name `contentAreaFilled` Data type boolean Flags expert
Border Painted?	Should the border of this button be displayed?

© 2011 Inductive Automation

Scripting name `borderPainted`
Data type boolean
Flags expert

Rollover? — If true, the button may indicate that the mouse is hovering over it.

Scripting name `rolloverEnabled`
Data type boolean
Flags expert

Label — Text displayed on this button.

Scripting name `text`
Data type String
Flags bindable

Image Path — The relative path of the image.

Scripting name `path`
Data type String
Flags bindable

Selected Image Path — The relative path of the image to be displayed when this component is selected (toggled on).

Scripting name `selectedPath`
Data type String

Styles — Contains the component's styles

Scripting name `styles`
Data type Dataset
Flags bindable | expert

Behavior

Focusable — If a button is not focusable, you will not be able to interact with it with the keyboard. This means you can't "tab" over to it.

Scripting name `focusable`
Data type boolean
Flags expert

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Enabled — If disabled, a component cannot be used.

Scripting name `enabled`
Data type boolean

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`

© 2011 Inductive Automation

Data type String

Cursor
The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values
0 Default
1 Crosshair
2 Text
3 Wait
12 Hand
13 Move
4 SW Resize
5 SE Resize
6 NW Resize
7 NE Resize
8 N Resize
9 S Resize
10 W Resize
11 E Resize

Data

Selected
State of this tToggle button.

Scripting name `selected`
Data type boolean
Flags bindable

Data Quality
The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Layout

Margin
The space between a button's text and its borders.

Scripting name `margin`
Data type Insets
Flags expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

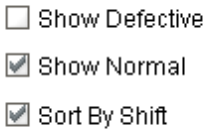
mouse
mouseMotion
item
action
focus
propertyChange
key

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.2.7 Check Box



Description

A CheckBox is a familiar component that represents a bit - it is either on (selected) or off (not selected). It is functionally equivalent to the Toggle Button component.

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the component.

Scripting name	`background`
Data type	Color

Text — The text displayed on the checkbox.

Scripting name	`text`
Data type	String
Flags	bindable

Margin — The internal margin that provides padding for the contents of this button.

Scripting name	`margin`
Data type	Insets
Flags	expert

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Rollover — If true, the button may indicate that the mouse is hovering over it.

Scripting name	`rolloverEnabled`
Data type	boolean
Flags	expert

Focusable — If a button is not focusable, you will not be able to interact with it with the keyboard. This means you can't "tab" over to it.

Scripting name	`focusable`
Data type	boolean
Flags	expert

Common

© 2011 Inductive Automation

Name
The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled
If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible
If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Opaque
If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name	`opaque`
Data type	boolean

Cursor
The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default
	1 Crosshair
	2 Text
	3 Wait
	12 Hand
	13 Move
	4 SW Resize
	5 SE Resize
	6 NW Resize
	7 NE Resize
	8 N Resize
	9 S Resize
	10 W Resize
	11 E Resize

Data

Selected
The current state of the checkbox.

Scripting name	`selected`
Data type	boolean
Flags	bindable

Data Quality
The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Layout

© 2011 Inductive Automation

Horizontal Alignment — The horizontal alignment of the button's contents (text and/or image)

Scripting name	`horizontalAlignment`
Data type	int
Values	`2` Left `0` Center `4` Right `10` Leading `11` Trailing

Vertical Alignment — The vertical alignment of the button's contents (text and/or image)

Scripting name	`verticalAlignment`
Data type	int
Values	`1` Top `0` Center `3` Bottom

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
item
action
focus
propertyChange
key

Scripting Functions

This component has no special scripting functions.

7.2.8 Radio Button

Description

The radio button is similar to the CheckBox component, except for one special property. All radio buttons in the same Container (including the Root Container) will automatically be mutually exclusive. This means that only one radio button can be selected at a time. Radio buttons are a good way to let the user choose just one of a number of options. Dropdown Lists are another good way to do this.

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`

© 2011 Inductive Automation

Data type Color

Background Color — The background color of the component.

Scripting name background
Data type Color

Text — Text of this component

Scripting name text
Data type String
Flags bindable

Margin — The internal margin that provides padding for the contents of this button.

Scripting name margin
Data type Insets
Flags expert

Styles — Contains the component's styles

Scripting name styles
Data type Dataset
Flags bindable | expert

Behavior

Rollover — If true, the button may indicate that the mouse is hovering over it.

Scripting name rolloverEnabled
Data type boolean
Flags expert

Focusable — If a button is not focusable, you will not be able to interact with it with the keyboard. This means you can't "tab" over to it.

Scripting name focusable
Data type boolean
Flags expert

Common

Name — The name of this component.

Scripting name name
Data type String
Flags bindable

Enabled — If disabled, a component cannot be used.

Scripting name enabled
Data type boolean

Visible — If disabled, the component will be hidden.

Scripting name visible
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name border
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name toolTipText
Data type String

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.

© 2011 Inductive Automation

Scripting name `opaque`
Data type boolean

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values
- 0 Default
- 1 Crosshair
- 2 Text
- 3 Wait
- 12 Hand
- 13 Move
- 4 SW Resize
- 5 SE Resize
- 6 NW Resize
- 7 NE Resize
- 8 N Resize
- 9 S Resize
- 10 W Resize
- 11 E Resize

Data

Selected — The current state of the RadioButton.

Scripting name `selected`
Data type boolean
Flags bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Layout

Horizontal Alignment — The horizontal alignment of the button's contents (text and/or image)

Scripting name `horizontalAlignment`
Data type int
Values
- 2 Left
- 0 Center
- 4 Right
- 10 Leading
- 11 Trailing

Vertical Alignment — The vertical alignment of the button's contents (text and/or image)

Scripting name `verticalAlignment`
Data type int
Values
- 1 Top
- 0 Center
- 3 Bottom

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

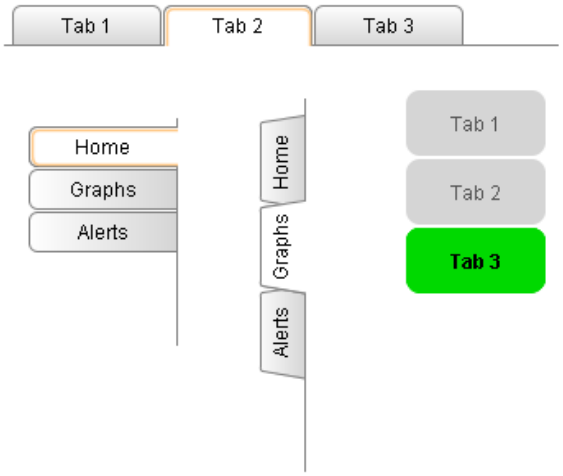
- mouse
- mouseMotion
- item
- action
- focus

© 2011 Inductive Automation

propertyChange
key

Scripting Functions
This component has no special scripting functions.

7.2.9 Tab Strip



Tab strips are highly customizable.

Description

In general, a tab strip is just a single-selection multiple choice component. In practice it is used anywhere that a user needs to be able to select between multiple windows or screens. It is most commonly used in a docked window to provide automatic window navigation. To support this typical use-case, the tab strip has two navigation modes:

1. **Swap to Window.** *(default)* The tab strip will automatically call `system.nav.swapTo()` with the name of the selected tab. This facilitates very easy navigation for most common projects.
2. **Disabled.** The tab strip doesn't do anything when the tab selection changes. Users can implement their own via property bindings or by responding to the `propertyChange` scripting event.

The tab strips visual style is highly customizable. There are different rendering styles, and things such as fonts, colors, line thicknesses, hover colors, and gradients are customizable within each rendering style. Use the Tab Strip's customizer to come up with a style that suits your project, as well as to manage the tabs that are present. The tabs and their styles are all stored in a dataset property (called *Tab Data*), so they can be modified at runtime as well.

See also:
Typical Navigation Strategy

Properties

Appearance

Background Color — The background color of the component.
Scripting name `background`

Data type Color

Orientation

Orientation of the tab strip.

Scripting name `orientation`
Data type int
Values
0 Top
1 Left
2 Bottom
3 Right

Selected Tab

Name of the selected tab. This is also the name of the window that, if it exists, will be swapped to when this tab is pressed.

Scripting name `selectedTab`
Data type String
Flags bindable

Renderer

The renderer to use when rendering tabs.

Scripting name `renderer`
Data type int
Values
0 Simple
1 Fancy
2 Folder

Size Mode

The sizing mode tabs use when deciding their size. Automatic means every tab is the same fixed size. Individual lets each tab decide its own size based on the size of its text.

Scripting name `sizeMode`
Data type int
Values
0 Automatic
1 Individual

Intertab Space

The amount of space between each tab.

Scripting name `interTabSpace`
Data type int

Text Padding

Padding on each side of the text inside a tab.

Scripting name `textPadding`
Data type int

Rounding Radius

Rounding radius for the tab corners.

Scripting name `roundingRadius`
Data type int

Separator Thickness

Thickness of the line drawn across the bottom and around each tab.

Scripting name `separatorThickness`
Data type float

Separator Color

Color of the line drawn across the bottom and around each tab.

Scripting name `separatorColor`
Data type Color

Antialias

Draw with antialias on? Makes text smoother

Scripting name `antialias`
Data type boolean
Flags expert

Styles

Contains the component's styles

Scripting name `styles`
Data type Dataset

© 2011 Inductive Automation

Flags bindable | expert

Behavior

Navigation Mode

Navigation mode. Disabled does nothing when a tab is pressed. Swap to window swaps to the window whose name corresponds to the name of the selected tab, provided that window exists.

Scripting name `navigationMode`
Data type int
Values
`0` Disabled
`1` Swap to window

Common

Name

The name of this component.

Scripting name `name`
Data type String
Flags bindable

Enabled

If disabled, a component cannot be used.

Scripting name `enabled`
Data type boolean

Visible

If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Cursor

The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values
`0` Default
`1` Crosshair
`2` Text
`3` Wait
`12` Hand
`13` Move
`4` SW Resize
`5` SE Resize
`6` NW Resize
`7` NE Resize
`8` N Resize
`9` S Resize
`10` W Resize
`11` E Resize

Data

Tab Data

Tab Data.

Scripting name `tabData`
Data type Dataset

Data Quality

The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

© 2011 Inductive Automation

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.3 Display

7.3.1 Label

Description

The Label is one of the most versatile components. It can display text, images, or both. Its text can be HTML formatted (like most components). It can even be made to respond to user interaction through its events.

Labels are one of the most common components that you will want to add *dynamic properties* to. For instance, you can put an integer dynamic property "state" on a label, and then bind the text to be "On" when the state=1 and "Off" otherwise, using an expression binding. Bind the background color to be red when the state is 0, and green when the state is 1 using a property binding. Now you have a re-usable binary state indicator. While you could have used the Multi-State Indicator to achieve the same effect, the exercise is good practice for creating custom components. You can see how the flexibility of bindings and dynamic properties make the Label extremely versatile.

See also:
Dynamic Properties
Property Bindings

Properties

Appearance

Font	Font of text of this component

© 2011 Inductive Automation

Scripting name	`font`
Data type	Font

Foreground Color — The color of the Label's text.

Scripting name	`foreground`
Data type	Color
Flags	bindable

Background Color — The background color of the label, if opaque is set to "true".

Scripting name	`background`
Data type	Color
Flags	bindable

Image Path — The relative path of the image.

Scripting name	`path`
Data type	String
Flags	bindable

Disabled Image Path — The relative path of the image to be displayed when this component is not enabled.

Scripting name	`disabledPath`
Data type	String
Flags	expert

Icon-Text Spacing — The space (in pixels) between the icon (if any) and the text (if any)

Scripting name	`iconTextGap`
Data type	int

Rotation — The angle of rotation in degrees.

Scripting name	`rotation`
Data type	int

Antialias — Draw with antialias on? Makes text smoother

Scripting name	`antialias`
Data type	boolean
Flags	expert

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

© 2011 Inductive Automation

Scripting name `border`
Data type Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name `toolTipText`
Data type String

Opaque
If false, backgrounds are not drawn. If true, backgrounds are drawn.
Scripting name `opaque`
Data type boolean

Cursor
The mouse cursor to use when hovering over this component.
Scripting name `cursorCode`
Data type int
Values
`0` Default
`1` Crosshair
`2` Text
`3` Wait
`12` Hand
`13` Move
`4` SW Resize
`5` SE Resize
`6` NW Resize
`7` NE Resize
`8` N Resize
`9` S Resize
`10` W Resize
`11` E Resize

Data

Text
Text of this Label
Scripting name `text`
Data type String
Flags bindable

Data Quality
The data quality code for any tag bindings on this component.
Scripting name `dataQuality`
Data type int
Flags bindable | expert

Layout

Horizontal Alignment
Determines the alignment of the label's contents along the X axis
Scripting name `horizontalAlignment`
Data type int
Values
`2` Left
`0` Center
`4` Right
`10` Leading
`11` Trailing

Horizontal Text Position
Determines the horizontal position of the label's text, relative to its image
Scripting name `horizontalTextPosition`
Data type int
Values
`2` Left
`0` Center
`4` Right
`10` Leading

© 2011 Inductive Automation

11 Trailing

Vertical Alignment — Determines the alignment of the label's contents along the Y axis

Scripting name	`verticalAlignment`
Data type	int
Values	1 Top 0 Center 3 Bottom

Vertical Text Position — Determines the vertical position of the label's text, relative to its image

Scripting name	`verticalTextPosition`
Data type	int
Values	1 Top 0 Center 3 Bottom

Scripting

Events

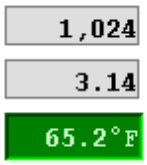
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.3.2 Numeric Label



Description

This component is a specialized label designed to display a number. It can include units, and has an integrated number format string. By default the number is displayed bold and the units are not. This can be customized, see the Prefix and Suffix expert properties. This label's text is constructed as follows:

```
Prefix + numberFormat(Value, Pattern) + Suffix + Units
```

It is important to note that you could customize the standard Label component using custom properties and bindings to mimic this component exactly. If this component doesn't do something that you need, you can make your own numeric label and use it everywhere in your project.

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

© 2011 Inductive Automation

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color
Flags	bindable

Background Color — The background color of the component.

Scripting name	`background`
Data type	Color
Flags	bindable

Number Format Pattern — The number formatting string used to format the value.

Scripting name	`pattern`
Data type	String

Image Path — The relative path of the image.

Scripting name	`path`
Data type	String
Flags	bindable

Disabled Image Path — The relative path of the image to be displayed when this component is not enabled.

Scripting name	`disabledPath`
Data type	String
Flags	expert

Icon-Text Spacing — The space (in pixels) between the icon (if any) and the text (if any)

Scripting name	`iconTextGap`
Data type	int

Rotation — The angle of rotation in degrees.

Scripting name	`rotation`
Data type	int

Antialias — Draw with antialias on? Makes text smoother

Scripting name	`antialias`
Data type	boolean
Flags	expert

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

© 2011 Inductive Automation

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Opaque
If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name	`opaque`
Data type	boolean

Cursor
The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default **1** Crosshair **2** Text **3** Wait **12** Hand **13** Move **4** SW Resize **5** SE Resize **6** NW Resize **7** NE Resize **8** N Resize **9** S Resize **10** W Resize **11** E Resize

Data

Value
The numeric value of this label.

Scripting name	`value`
Data type	double
Flags	bindable

Units
The engineering units to display after the number.

Scripting name	`units`
Data type	String

Prefix
A string that will be placed before the number.

Scripting name	`prefix`
Data type	String
Flags	expert

Suffix
A string that will be placed after the number, and before the units.

Scripting name	`suffix`
Data type	String
Flags	expert

Data Quality
The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Layout

© 2011 Inductive Automation

Horizontal Alignment — Determines the alignment of the label's contents along the X axis

Scripting name	`horizontalAlignment`
Data type	int
Values	**2** Left **0** Center **4** Right **10** Leading **11** Trailing

Horizontal Text Position — Determines the horizontal position of the label's text, relative to its image

Scripting name	`horizontalTextPosition`
Data type	int
Flags	expert
Values	**2** Left **0** Center **4** Right **10** Leading **11** Trailing

Vertical Alignment — Determines the alignment of the label's contents along the Y axis

Scripting name	`verticalAlignment`
Data type	int
Values	**1** Top **0** Center **3** Bottom

Vertical Text Position — Determines the vertical position of the label's text, relative to its image

Scripting name	`verticalTextPosition`
Data type	int
Flags	expert
Values	**1** Top **0** Center **3** Bottom

Uncategorized

Text — Text of this Label

Scripting name	`text`
Data type	String
Flags	bindable | read-only

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.3.3 Multi-State Indicator

Description

This component is a specialized label used to display a discrete state. The state must be represented by an integer, but the values and number of different states is customizable. Use the component's styles customizer to configure the different states.

See also:
Component Styles

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color
Flags	bindable

Background Color — The background color of the component.

Scripting name	`background`
Data type	Color
Flags	bindable

Image Path — The relative path of the image.

Scripting name	`path`
Data type	String
Flags	bindable

Disabled Image Path — The relative path of the image to be displayed when this component is not enabled.

Scripting name	`disabledPath`
Data type	String
Flags	expert

Icon-Text Spacing — The space (in pixels) between the icon (if any) and the text (if any)

Scripting name	`iconTextGap`
Data type	int

Rotation — The angle of rotation in degrees.

Scripting name	`rotation`
Data type	int

Antialias — Draw with antialias on? Makes text smoother

Scripting name	`antialias`
Data type	boolean
Flags	expert

Styles — Contains the component's styles

© 2011 Inductive Automation

Scripting name `styles`
Data type Dataset
Flags bindable | expert

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Enabled — If disabled, a component cannot be used.

Scripting name `enabled`
Data type boolean

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name `opaque`
Data type boolean

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

State — The current state of the component.

Scripting name `state`
Data type int
Flags bindable

Text — Text of this Label

Scripting name `text`

© 2011 Inductive Automation

Data type String
Flags bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Layout

Horizontal Alignment — Determines the alignment of the label's contents along the X axis

Scripting name `horizontalAlignment`
Data type int
Values
- `2` Left
- `0` Center
- `4` Right
- `10` Leading
- `11` Trailing

Horizontal Text Position — Determines the horizontal position of the label's text, relative to its image

Scripting name `horizontalTextPosition`
Data type int
Flags expert
Values
- `2` Left
- `0` Center
- `4` Right
- `10` Leading
- `11` Trailing

Vertical Alignment — Determines the alignment of the label's contents along the Y axis

Scripting name `verticalAlignment`
Data type int
Values
- `1` Top
- `0` Center
- `3` Bottom

Vertical Text Position — Determines the vertical position of the label's text, relative to its image

Scripting name `verticalTextPosition`
Data type int
Flags expert
Values
- `1` Top
- `0` Center
- `3` Bottom

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.3.4 LED Display

Description

The LED display is a stylized numeric or alphanumeric label. It has three different visual styles which all correspond to a kind of physical display: 7-segment, 14-segment, and 5x7 matrix. By default this component is in numeric mode, which means you should use its *Value* property. If you need to display characters as well, switch the mode to alphanumeric, and use the *Text* property.

Properties

Appearance

Style — The visual style of the display.

Scripting name	`style`
Data type	int
Values	**7** 7 Segment **14** 14 Segment **34** 5x7 Matrix

Background Color — The color of the background

Scripting name	`background`
Data type	Color

LED Lit — The color of lit LED segments

Scripting name	`glyphForeground`
Data type	Color

LED Unlit — The color of unlit LED segments

Scripting name	`glyphBackground`
Data type	Color

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Mode — The mode of the display.

Scripting name	`mode`
Data type	int
Values	**0** Numeric **1** Alphanumeric

Number Format Pattern — The number formatting string used to format the value.

Scripting name	`numberFormat`
Data type	String

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

© 2011 Inductive Automation

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default
	1 Crosshair
	2 Text
	3 Wait
	12 Hand
	13 Move
	4 SW Resize
	5 SE Resize
	6 NW Resize
	7 NE Resize
	8 N Resize
	9 S Resize
	10 W Resize
	11 E Resize

Data

Value — The numeric value of the display, used when Mode is Numeric

Scripting name	`value`
Data type	double
Flags	bindable

Text — The text value of the display, used when Mode is Alphanumeric

Scripting name	`text`
Data type	String
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Layout

Horizontal Alignment — Determines the alignment of the display's contents along the X axis

Scripting name	`horizontalAlignment`
Data type	int
Values	**2** Left
	0 Center
	4 Right

Letter Gap — The percentage of the height to be used as an inter-character spacing

© 2011 Inductive Automation

Scripting name	`gap`
Data type	float
Flags	expert

Margin — The margin for the interior of the display

Scripting name	`margin`
Data type	Insets
Flags	expert

Scripting

Events

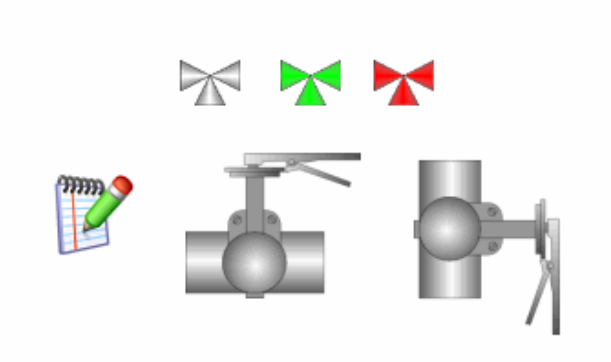
The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- propertyChange

Scripting Functions

This component has no special scripting functions.

7.3.5 Image



Description

The image component is a deceptively powerful component. While you can use other components, like the Label, to display images as well, this component gives you much more flexibility. In particular, this component has 4 important features for displaying images:

1. **Scaling.**
2. **Rotation.** You can use rotation to create spinning animations by binding it to a Timer component.
3. **Color Tinting.** Dynamically apply a color tint to an image to allow it to display realtime status.
4. **Color Swapping**. Use color swapping to change one specific color in an image to another, on the fly. Also used for realtime status display.

To choose an image, simply press the browse button ( ) next to this component's *Image Path* property. You can drag new images (*.png, *.gif, *.jpg, *.bmp) into the Image Management window to upload them.

Images are stored on the Gateway, *not* in your window or project. This means that you can alter an image globally, and it will affect all windows in all projects. It also means that you must be careful to migrate custom images if you do project backups (as opposed to Gateway backups, which will automatically include both projects and images)

Properties

© 2011 Inductive Automation

Appearance

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default **1** Crosshair **2** Text **3** Wait **12** Hand **13** Move **4** SW Resize **5** SE Resize **6** NW Resize **7** NE Resize **8** N Resize **9** S Resize **10** W Resize **11** E Resize

Data

Image Path — The relative path of the image.

Scripting name	`path`
Data type	String
Flags	bindable

Disabled Image Path — The relative path of the image to be displayed when this component is not enabled.

© 2011 Inductive Automation

Scripting name	`disabledPath`
Data type	String
Flags	expert

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Image Manipulation

Stretch Mode — Sets the stretch mode for this image.

Scripting name	`stretchMode`
Data type	int
Values	0 No Stretch 1 Bounds 3 % Bounds 2 Parameters

Stretch Width — If stretch mode is "Parameters", this will be the stretched width of the image
If stretch mode is "% Bounds", this will be the percentage of the component's width.

Scripting name	`stretchWidth`
Data type	int
Flags	bindable

Stretch Height — If stretch mode is "Parameters", this will be the stretched height of the image
If stretch mode is "% Bounds", this will be the percentage of the component's height.

Scripting name	`stretchHeight`
Data type	int
Flags	bindable

Color Swap Filter — Swap a specific color to another.

Scripting name	`useColorSwap`
Data type	boolean
Flags	bindable

Swap From — If the Color Swap Filter is on, this color will be changed to the Swap To color.

Scripting name	`swapFromColor`
Data type	Color
Flags	bindable

Swap To — If the Color Swap Filter is on, the Swap From color will be changed to this color.

Scripting name	`swapToColor`
Data type	Color
Flags	bindable

Swap Threshold — Threshold (0-255) for the swap from color matching. 0 is no tolerance, 255 is max tolerance.

Scripting name	`swapThreshold`
Data type	int
Flags	expert

Tint Filter — Tint the entire image a color (works best with greyscale images)

© 2011 Inductive Automation

Scripting name	`useTint`
Data type	boolean
Flags	bindable

Tint Color — If the Tint Filter is on, this is the color of the tint.

Scripting name	`tintColor`
Data type	Color
Flags	bindable

Flip Horizontal — Flip (mirror) the image horizontally.

Scripting name	`flipHorizontal`
Data type	boolean

Flip Vertical — Flip (mirror) the image vertically.

Scripting name	`flipVertical`
Data type	boolean

Rotation — The angle of rotation in degrees.

Scripting name	`rotation`
Data type	int

Scripting

Events

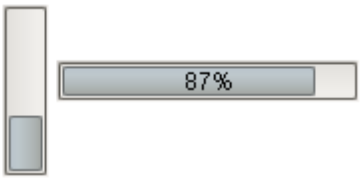
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.3.6 Progress Bar



Description

Visually indicates the progress of some task. Can be used to display any value that has an upper and lower bound.

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the component.

© 2011 Inductive Automation

Scripting name	`background`
Data type	Color

Horizontal?

If true, the progress bar will display horizontally, else it will display vertically

Scripting name	`horizontal`
Data type	boolean

Show Percentage?

If true, the progress bar will display its percentage.

Scripting name	`stringPainted`
Data type	boolean

Direction

Determines the direction of progress for this progress bar.

Scripting name	`direction`
Data type	int
Flags	expert
Values	`0` Left to Right
	`1` Right to Left

Styles

Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Indeterminate?

When true, the progressbar displays animation indicating that something is happening, but it will take an indeterminate amount of time

Scripting name	`indeterminate`
Data type	boolean
Flags	bindable

Common

Name

The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled

If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible

If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Opaque

If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name	`opaque`
Data type	boolean

© 2011 Inductive Automation

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	0 Default 1 Crosshair 2 Text 3 Wait 12 Hand 13 Move 4 SW Resize 5 SE Resize 6 NW Resize 7 NE Resize 8 N Resize 9 S Resize 10 W Resize 11 E Resize

Data

Value — The current state of the Progress Bar.

Scripting name	`value`
Data type	int
Flags	bindable

Maximum — The maximum value that this progress bar will reach

Scripting name	`maximum`
Data type	int
Flags	bindable

Minimum — The minimum value that this progress bar will reach

Scripting name	`minimum`
Data type	int
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

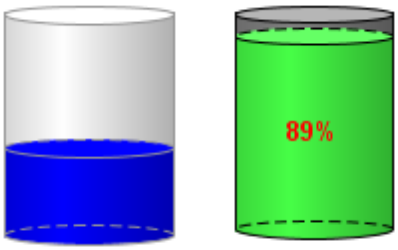
mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.3.7 Cylindrical Tank



Description

A component that looks like a 3D cylindrical tank, with some liquid inside. The liquid rises and falls as the *Value* property changes.

Properties

Appearance

Font
Font of text of this component
Scripting name `font`
Data type Font

Foreground Color
The foreground color of the component.
Scripting name `foreground`
Data type Color

Background Color
The background color of the component.
Scripting name `background`
Data type Color

Rotation
The angle of rotation in degrees.
Scripting name `rotation`
Data type int

Anti Alias
Draw component using anti-aliasing?
Scripting name `antiAlias`
Data type boolean
Flags expert

Units
Units of measure for tank contents
Scripting name `units`
Data type String

Show Value
Show numeric value, capacity, and units?
Scripting name `showValue`
Data type boolean

Show Percentage
Show percentage of tank filled?
Scripting name `showPercent`
Data type boolean

Font Color
The color of the value and/or percentage labels.
Scripting name `fontColor`
Data type Color

© 2011 Inductive Automation

Tank Color — Color of the non-filled tank section

Scripting name	`tankColor`
Data type	Color
Flags	bindable

Liquid Color — Color of the filled tank section

Scripting name	`liquidColor`
Data type	Color
Flags	bindable

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default **1** Crosshair **2** Text **3** Wait **12** Hand **13** Move **4** SW Resize **5** SE Resize **6** NW Resize **7** NE Resize **8** N Resize **9** S Resize **10** W Resize **11** E Resize

Data

© 2011 Inductive Automation

Value — Numeric value of tank's level

Scripting name	`value`
Data type	int
Flags	bindable

Capacity — Total capacity of tank

Scripting name	`capacity`
Data type	int
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

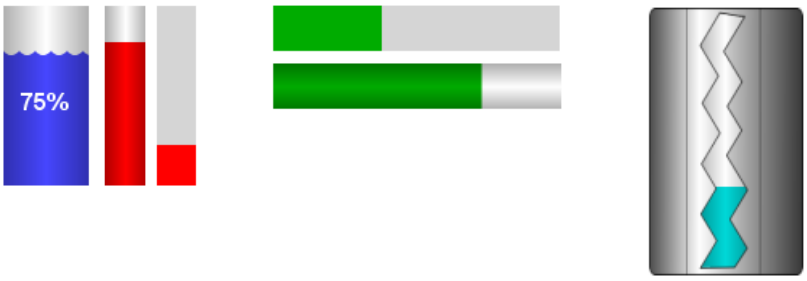
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.3.8 Level Indicator



Description

A component that displays the level of fullness of some container. This is basically a visually simplified version of the Cylindrical Tank component. By turning on and off the Gradient and Liquid Waves properties, you can control how fancy this component looks. This component is well suited to be put behind images of tanks with transparent cutaways.

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Units — Units of measure for tank contents

Scripting name	`units`
Data type	String

© 2011 Inductive Automation

Show Value

Show numeric value, capacity, and units?

Scripting name `showValue`
Data type boolean

Show Percentage

Show percentage of tank filled?

Scripting name `showPercent`
Data type boolean

Orientation

Determines which way the level "grows" for an increase in value

Scripting name `orientation`
Data type int
Values
0 Bottom to Top
1 Left to Right
2 Top to Bottom
3 Right to Left

Filled Color

Set the color of filled portion.

Scripting name `foreground`
Data type Color

Background Color

The color of the background.

Scripting name `background`
Data type Color

Gradient

Draw level as 3D gradient?.

Scripting name `gradient`
Data type boolean

Liquid Waves

Draw liquid 'waves'?

Scripting name `waves`
Data type boolean

Wave Length

The length of each 'wave'

Scripting name `waveLength`
Data type int
Flags expert

Wave Height

The height of each 'wave'

Scripting name `waveHeight`
Data type int
Flags expert

Font Color

Scripting name `fontColor`
Data type Color

Anti Alias

Draw component using anti-aliasing?

Scripting name `antiAlias`
Data type boolean
Flags expert

Styles

Contains the component's styles

Scripting name `styles`
Data type Dataset
Flags bindable | expert

Common

© 2011 Inductive Automation

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default **1** Crosshair **2** Text **3** Wait **12** Hand **13** Move **4** SW Resize **5** SE Resize **6** NW Resize **7** NE Resize **8** N Resize **9** S Resize **10** W Resize **11** E Resize

Data

Value — Numeric value of tank's level

Scripting name	`value`
Data type	int
Flags	bindable

Capacity — Total capacity of tank

Scripting name	`capacity`
Data type	int
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

© 2011 Inductive Automation

Scripting

Events

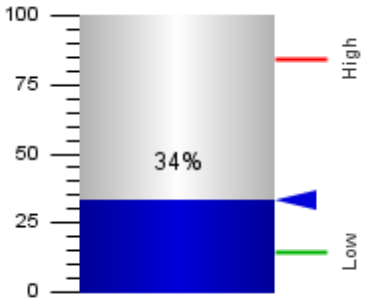
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.3.9 Linear Scale



Two linear scales flanking a level indicator

Description

The linear scale component has two main purposes. The first is to display a series of tick marks and labels that visually represent a linear range between a minimum value and a maximum value. The second purpose is to display indicators that represent a value or range of values, correctly positioned in on the linear scale. In the example above, two linear scales are used to flank a level indicator. The scale on the left has only tick marks, and no indicators. The scale on the right is used to display three indicators and no tick marks.

To configure the indicators, you use the Linea Scale's "Scale Indicators" customizer. To configure the tick marks, you use the scale's various properties that determine the minimum value, maximum value, and the various tick mark spans.

Properties

Appearance

Mirror	Mirror the scale so it paints against the opposite edge. Scripting name `mirror` Data type boolean
Reverse Range	Reverse the scale so that values go from high to low instead of low to high. Scripting name `reverseRange` Data type boolean
Label Angle	Changes the angle that the labels are drawn

© 2011 Inductive Automation

Scripting name `labelAngle`
Data type int
Values
`0` Right
`90` Down
`180` Left
`270` Up

Margin
The margin to leave blank as a percentage of the total height or width of the scale.
Scripting name `margin`
Data type double

Major Tick Length
The line length for major ticks, in pixels.
Scripting name `majorTickLength`
Data type double

Major Tick Thickness
The line thickness for major ticks, in pixels.
Scripting name `majorTickStroke`
Data type float

Major Tick Color
The line color for major ticks
Scripting name `majorTickColor`
Data type Color

Label Format
The label format string. Examples: "%.1f" will render numbers like "15.0", "%.0f" will render numbers like "15". Using the empty string "" will disable the labels.
Scripting name `majorTickLabelFormat`
Data type String

Label Font
The font used for drawing tick labels.
Scripting name `majorTickFont`
Data type Font

Label Color
The color used for drawing tick labels.
Scripting name `majorTickLabelColor`
Data type Color

Minor Tick Length
The line length for minor ticks, in pixels.
Scripting name `minorTickLength`
Data type double

Minor Tick Thickness
The line thickness for minor ticks, in pixels
Scripting name `minorTickStroke`
Data type float

Minor Tick Color
The line color for minor ticks.
Scripting name `minorTickColor`
Data type Color

Fine Tick Length
The line length for fine ticks, in pixels.
Scripting name `fineTickLength`
Data type double

Fine Tick Thickness
The line thickness for fine ticks, in pixels
Scripting name `fineTickStroke`
Data type float

© 2011 Inductive Automation

Fine Tick Color

The line color for fine ticks.

Scripting name	`fineTickColor`
Data type	Color

Common

Name

The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled

If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible

If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor

The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default
	1 Crosshair
	2 Text
	3 Wait
	12 Hand
	13 Move
	4 SW Resize
	5 SE Resize
	6 NW Resize
	7 NE Resize
	8 N Resize
	9 S Resize
	10 W Resize
	11 E Resize

Data

Min Value

The lower bound of the scale

Scripting name	`minValue`
Data type	double
Flags	bindable

Max Value

The upper bound of the scale

Scripting name	`maxValue`
Data type	double
Flags	bindable

© 2011 Inductive Automation

Major Tick Span — The span length for major ticks. Should be a multiple of the minor and fine tick spans.
Scripting name: majorTickSpan
Data type: double

Minor Tick Span — The span length for minor ticks. Should be a factor of the major tick span and a multiple of the fine tick spans. Use zero to disable minor ticks.
Scripting name: minorTickSpan
Data type: double

Fine Tick Span — The span length for fine ticks. Should be a factor of the major and minor tick spans. Use zero to disable fine ticks.
Scripting name: fineTickSpan
Data type: double

Indicators — This dataset stores the indicators (if any) for the scale.
Scripting name: indicators
Data type: Dataset

Data Quality — The data quality code for any tag bindings on this component.
Scripting name: dataQuality
Data type: int
Flags: bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.3.10 Barcode

Description

The barcode component displays some text as a barcode. The supported formats are:

Code 128
Code 39
Extended Code 39
Codabar
Interleaved Code 25
MSI
EAN-13
EAN-8

© 2011 Inductive Automation

See also:
system.print.createPrintJob

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the component.

Scripting name	`background`
Data type	Color

Barcode Background — The background color of the actual barcode

Scripting name	`barcodeBackground`
Data type	Color

Show Text? — If true, the code is displayed in human-readable text beneath the barcode

Scripting name	`showText`
Data type	boolean

Barcode Height — The height of the barcode

Scripting name	`barcodeHeight`
Data type	int

Narrowest Bar Width — The width (in pixels) of the narrowest bar.

Scripting name	`narrowestBarWidth`
Data type	int

Rotation — The angle of rotation in degrees.

Scripting name	`angleDegrees`
Data type	int
Flags	expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

© 2011 Inductive Automation

Scripting name	`toolTipText`
Data type	String

Data

Code — The code string that is converted into a barcode to display

Scripting name	`code`
Data type	String
Flags	bindable

Barcode Format — The barcode format to display.

Scripting name	`barcodeType`
Data type	int
Values	`0` Code 39 `1` Code 39 (narrow) `2` Extended Code 39 `3` Extended Code 39 (narrow) `4` Code 128 `5` Codabar `6` Codabar (narrow) `7` Interleaved Code 25 `8` Interleaved Code 25 (narrow) `9` MSI `10` EAN-13 `11` EAN-8

Check Digit — Include Check Digit?

Scripting name	`checkDigit`
Data type	boolean

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

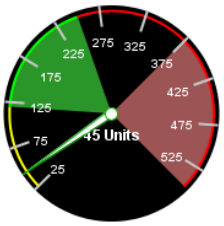
- mouse
- mouseMotion
- propertyChange

Scripting Functions

This component has no special scripting functions.

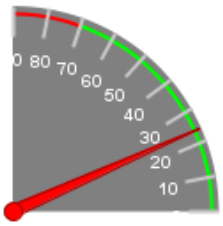
© 2011 Inductive Automation

7.3.11 Meter



Meters have customizable segments

A few gradient and transparent circles can give your meter some shine



Meters can be many shapes

Description

A meter display shows a value on a needle-gauge. The gauge's range can be broken up into five intervals. The intervals can have their own edge and background colors.

How the meter looks is affected by its appearance properties. You can modify colors, thicknesses, start and extend angles, needle size, etc to get the meter that you want. For example, the meter on the far right of the example has a Meter Angle Extent of 90°, a Meter Angle of 45°, a reversed range, and 2 intervals.

Properties

Appearance

Units
A string to describe the units for the current value label.
Scripting name `units`
Data type String

Dial Background
The background color of the dial face.
Scripting name `dialBackground`
Data type Color

Needle Color
The color of the meter's needle.
Scripting name `needleColor`
Data type Color

Needle Size
The size of the base of the needle.
Scripting name `needleSize`
Data type float

Needle Stroke Color
The color of the needle's stroke.
Scripting name `needleStrokeColor`
Data type Color

Needle Stroke Size
The size of the needle's stroke.
Scripting name `needleStrokeSize`
Data type float

Value Color
The color of the meter's current value label.
Scripting name `valueColor`
Data type Color

© 2011 Inductive Automation

Tick Label Color
The color of the tick labels
Scripting name `tickLabelColor`
Data type Color

Tick Color
The color of tick marks.
Scripting name `tickColor`
Data type Color

Tick Size
The distance between ticks.
Scripting name `tickSize`
Data type double

Show Tick Labels?
If true, value will be shown on interval-boundry ticks.
Scripting name `ticks`
Data type boolean

Value Label Font
The font to use for the current value label.
Scripting name `valueFont`
Data type Font

Tick Label Font
The font to use for the tick labels.
Scripting name `labelFont`
Data type Font

Value Format
The number format to use for the value label.
Scripting name `valueLabelFormat`
Data type String

Tick Format
The number format to use for the tick labels.
Scripting name `tickLabelFormat`
Data type String

Arc Width
The width of the colored interval arcs
Scripting name `arcWidth`
Data type float

Meter Angle Extent
The extent, in degrees, of the entire meter.
Scripting name `meterAngleExtent`
Data type int

Meter Angle
The angle in degrees of the centerpoint of the meter (90 is straight up).
Scripting name `meterAngle`
Data type int

Dial Shape
The shape of the dial. This property determines how the dial face looks in the area not covered by the meter angle extent.
Scripting name `dialType`
Data type int
Values
1 Chord
0 Circle
2 Pie

Styles
Contains the component's styles
Scripting name `styles`
Data type Dataset
Flags bindable | expert

Common

© 2011 Inductive Automation

Name
The name of this component.
Scripting name `name`
Data type String
Flags bindable

Visible
If disabled, the component will be hidden.
Scripting name `visible`
Data type boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name `border`
Data type Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name `toolTipText`
Data type String

Cursor
The mouse cursor to use when hovering over this component.
Scripting name `cursorCode`
Data type int
Values
`0` Default
`1` Crosshair
`2` Text
`3` Wait
`12` Hand
`13` Move
`4` SW Resize
`5` SE Resize
`6` NW Resize
`7` NE Resize
`8` N Resize
`9` S Resize
`10` W Resize
`11` E Resize

Data

Value
The value to display in this meter. The needle and current value label will change to reflect this.
Scripting name `value`
Data type double
Flags bindable

Overall Low Bound
The lower bound for the whole meter
Scripting name `overallLow`
Data type double
Flags bindable

Overall High Bound
The high bound for the whole meter
Scripting name `overallHigh`
Data type double
Flags bindable

Reverse Range?
If true, the meter will consider right to left needle movement as positive.
Scripting name `reverseRange`
Data type boolean

© 2011 Inductive Automation

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Intervals

Interval 1 Low — The lower bound of this interval.

Scripting name	`interval1Low`
Data type	double

Interval 1 High — The upper bound of this interval.

Scripting name	`interval1High`
Data type	double

Interval 1 Outline — The color to paint the arc of this interval.

Scripting name	`interval1Outline`
Data type	Color

Interval 1 Background — The color to fill the wedge of this interval.

Scripting name	`interval1Background`
Data type	Color

Interval 2 Low — The lower bound of this interval.

Scripting name	`interval2Low`
Data type	double

Interval 2 High — The upper bound of this interval.

Scripting name	`interval2High`
Data type	double

Interval 2 Outline — The color to paint the arc of this interval.

Scripting name	`interval2Outline`
Data type	Color

Interval 2 Background — The color to fill the wedge of this interval.

Scripting name	`interval2Background`
Data type	Color

Interval 3 Low — The lower bound of this interval.

Scripting name	`interval3Low`
Data type	double

Interval 3 High — The upper bound of this interval.

Scripting name	`interval3High`
Data type	double

Interval 3 Outline — The color to paint the arc of this interval.

Scripting name	`interval3Outline`
Data type	Color

Interval 3 Background — The color to fill the wedge of this interval.

Scripting name	`interval3Background`
Data type	Color

Interval 4 Low — The lower bound of this interval.

Scripting name	`interval4Low`
Data type	double

© 2011 Inductive Automation

Interval 4 High
The upper bound of this interval.
Scripting name `interval4High`
Data type double

Interval 4 Outline
The color to paint the arc of this interval.
Scripting name `interval4Outline`
Data type Color

Interval 4 Background
The color to fill the wedge of this interval.
Scripting name `interval4Background`
Data type Color

Interval 5 Low
The lower bound of this interval.
Scripting name `interval5Low`
Data type double

Interval 5 High
The upper bound of this interval.
Scripting name `interval5High`
Data type double

Interval 5 Outline
The color to paint the arc of this interval.
Scripting name `interval5Outline`
Data type Color

Interval 5 Background
The color to fill the wedge of this interval.
Scripting name `interval5Background`
Data type Color

Scripting

Events

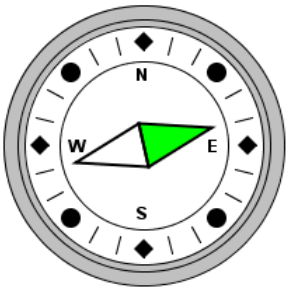
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

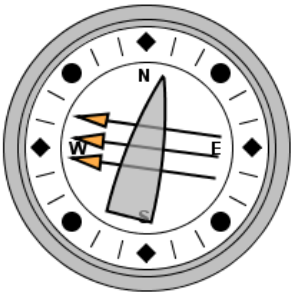
Scripting Functions

This component has no special scripting functions.

7.3.12 Compass



A basic compass.



Compasses can have up to 3 needles and have many needle types.

Description

The compass is a component that displays up to three needles at once on a cardinal direction compass. This can be useful for plotting anything that has a cardinal direction, such as the wind direction.

Each needle can be one of 9 different styles. Use the "Disabled" style to turn off any needle.

Properties

Appearance

Value 1 Color — The main color for Value 1's needle

Scripting name: `value1Color`
Data type: Color

Value 1 Outline — The outline color for value 1s needle

Scripting name: `value1OutlineColor`
Data type: Color

Value 2 Color — The main color for Value 2's needle

Scripting name: `value2Color`
Data type: Color

Value 2 Outline — The outline color for value 2s needle

Scripting name: `value2OutlineColor`
Data type: Color

Value 3 Color — The main color for Value 3's needle

Scripting name: `value3Color`
Data type: Color

Value 3 Outline — The outline color for value 3s needle

Scripting name: `value3OutlineColor`
Data type: Color

Label Font — The font to use for the compass's labels.

Scripting name: `labelFont`
Data type: Font

Rose Color — The background color of the rose.

Scripting name: `roseColor`
Data type: Color

Rose Highlight — The highlight color of the rose.

Scripting name: `roseHighlightColor`
Data type: Color

Center Color — The center color of the compass

Scripting name: `centerColor`
Data type: Color

Styles — Contains the component's styles

Scripting name: `styles`
Data type: Dataset
Flags: bindable | expert

Common

Name — The name of this component.

Scripting name: `name`

© 2011 Inductive Automation

Data type String
Flags bindable

Visible
If disabled, the component will be hidden.
Scripting name `visible`
Data type boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name `border`
Data type Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name `toolTipText`
Data type String

Cursor
The mouse cursor to use when hovering over this component.
Scripting name `cursorCode`
Data type int
Values
`0` Default
`1` Crosshair
`2` Text
`3` Wait
`12` Hand
`13` Move
`4` SW Resize
`5` SE Resize
`6` NW Resize
`7` NE Resize
`8` N Resize
`9` S Resize
`10` W Resize
`11` E Resize

Data

Value 1
Value 1 for the compass.
Scripting name `value1`
Data type double
Flags bindable

Value 1 Needle
The needle type for this vaule.
Scripting name `value1Needle`
Data type int
Values
`-1` Disabled
`0` Arrow
`1` Line
`2` Long
`3` Pin
`4` Plum
`5` Pointer
`6` Ship
`7` Wind
`9` Middle Pin

Value 2
Value 2 for the compass.
Scripting name `value2`
Data type double
Flags bindable

© 2011 Inductive Automation

Value 2 Needle — The needle type for this vaule.

Scripting name	`value2Needle`
Data type	int
Values	**-1** Disabled **0** Arrow **1** Line **2** Long **3** Pin **4** Plum **5** Pointer **6** Ship **7** Wind **9** Middle Pin

Value 3 — Value 3 for the compass.

Scripting name	`value3`
Data type	double
Flags	bindable

Value 3 Needle — The needle type for this vaule.

Scripting name	`value3Needle`
Data type	int
Values	**-1** Disabled **0** Arrow **1** Line **2** Long **3** Pin **4** Plum **5** Pointer **6** Ship **7** Wind **9** Middle Pin

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

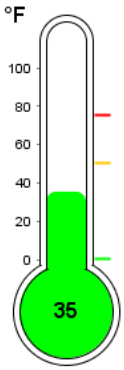
mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.3.13 Thermometer



Description

This component displays a temperature value depicted as a level in a mercury thermometer. Three temperature intervals can optionally be defined with their own colors. The mercury will change color based on the range that it is in.

Properties

Appearance

Units

A string to describe the units for the current value label.

Scripting name	`units`
Data type	int
Values	0 None 1 Fahrenheit 2 Celcius 3 Kelvin

Thermometer Color

The color of the outline of the thermometer.

Scripting name	`thermometerColor`
Data type	Color

Mercury Color

The default color of the mercury.

Scripting name	`mercuryColor`
Data type	Color

Value Color

The color of the meter's current value label.

Scripting name	`valueColor`
Data type	Color

Value Label Font

The font to use for the current value label.

Scripting name	`valueFont`
Data type	Font

Thermometer Width

The width of the lines used to draw the thermomoter

Scripting name	`strokeWidth`
Data type	int
Flags	expert

Use Range Color

Controls whether or not the mercury color changes based on the range it is in

Scripting name	`useSubrangePaint`
Data type	boolean

© 2011 Inductive Automation

Flags expert

Styles — Contains the component's styles

Scripting name `styles`
Data type Dataset
Flags bindable | expert

Behavior

Follow data in ranges — If true, the thermometer's Y axis will scale itself to zoom in on the current range.

Scripting name `followDataInSubranges`
Data type boolean
Flags expert

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values
- `0` Default
- `1` Crosshair
- `2` Text
- `3` Wait
- `12` Hand
- `13` Move
- `4` SW Resize
- `5` SE Resize
- `6` NW Resize
- `7` NE Resize
- `8` N Resize
- `9` S Resize
- `10` W Resize
- `11` E Resize

Data

Value — The value to display in this thermometer. The mercury level and value label will change to reflect this.

Scripting name `value`
Data type double
Flags bindable

© 2011 Inductive Automation

Overall Low Bound — The lower bound for the whole thermometer

Scripting name	`overallLow`
Data type	double
Flags	bindable

Overall High Bound — The high bound for the whole thermometer

Scripting name	`overallHigh`
Data type	double
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Intervals

Interval 1 Low — The lower bound of this interval.

Scripting name	`interval1Low`
Data type	double

Interval 1 High — The upper bound of this interval.

Scripting name	`interval1High`
Data type	double

Interval 1 Color — The color of this interval.

Scripting name	`interval1Color`
Data type	Color

Interval 2 Low — The lower bound of this interval.

Scripting name	`interval2Low`
Data type	double

Interval 2 High — The upper bound of this interval.

Scripting name	`interval2High`
Data type	double

Interval 2 Color — The color of this interval.

Scripting name	`interval2Color`
Data type	Color

Interval 3 Low — The lower bound of this interval.

Scripting name	`interval3Low`
Data type	double

Interval 3 High — The upper bound of this interval.

Scripting name	`interval3High`
Data type	double

Interval 3 Color — The color of this interval.

Scripting name	`interval3Color`
Data type	Color

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse

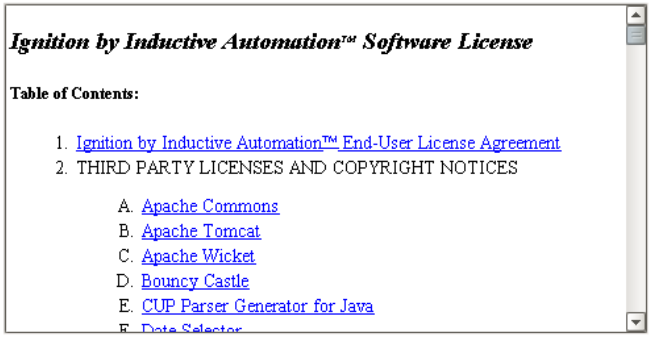
© 2011 Inductive Automation

mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.3.14 Document Viewer


Ignition by Inductive Automation™ Software License
Table of Contents:
1. Ignition by Inductive Automation™ End-User License Agreement
2. THIRD PARTY LICENSES AND COPYRIGHT NOTICES
A. Apache Commons
B. Apache Tomcat
C. Apache Wicket
D. Bouncy Castle
E. CUP Parser Generator for Java

Description

The document viewer is capable of loading and displaying a document that is available over the network at a URL. It is capable of displaying simple HTML and RTF documents. Although HTML links will be followed, it is *not* a fully functional interactive web browser. Its HTML support is rudimentary at best, and there is no JavaScript support. See the system.net.openURL function for a more robust solution for launching webpages, PDFs, etc.

This is component is useful for viewing machine manuals or operator protocol in HTML or RTF format. Note that in addition to HTML URLs (like "*http://www.google.com*"), you can load files as well using the URL format for files. Some examples:

```
file://localhost/C:/myfolder/file.txt
file://MyFileServer/resources/manuals/instructions.rtf"
```

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the component.

Scripting name	`background`
Data type	Color

Behavior

Link Action — What happens when the user clicks on a hpyerlink inside this document, if it is an HTML document.

Scripting name	`linkAction`
Data type	int
Values	`0` Launch Externally
	`1` Launch Internally

© 2011 Inductive Automation

2 Fire Event

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name	`opaque`
Data type	boolean

Data

Page URL — Set this to a URL to display that page. If the url startswith '/', it is assumed to be relative to the Gateway's HTTP address.

Scripting name	`page`
Data type	String
Flags	bindable

Content Type — The content type of this document. Exampl: text/html

Scripting name	`contentType`
Data type	String
Flags	expert

text — The text of the document. Should match the content type

Scripting name	`text`
Data type	String
Flags	expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- hyperlink
- focus
- propertyChange
- key

© 2011 Inductive Automation

Scripting Functions

This component has no special scripting functions.

7.3.15 IP Camera Viewer

Description

The IP camera viewing component displays a video stream from a network camera directly in one of your windows. This can be a very powerful tool for allowing operators to view remote or inaccessible locations. Cameras can provide positive feedback about the state and position of machinery, weather, and other factors.

This component is capable of displaying two types of video:

MJPEG (a.k.a. Motion JPEG) is a streaming video protocol that compresses video frames using standard JPEG compression. Compression rates are quite good, requiring low network bandwidth utilization. Framerates depend greatly on the dimensions of the video, but typically range from 1-20 frames per second.

JPEG stills is not a true video protocol, but is rather the practice of continually refreshing an image that a camera is constantly overwriting. Its simplicity means that many cameras support it (usually along with another protocol). Frame rates are typically lower than MJPEG because a new connection must be opened for each frame.

Most network cameras on the market support one, if not both of these protocols. Even better, if you have an existing CCTV camera system, **video server** devices are available that CCTV camera inputs and provide MJPEG streams the network.

Finding the URL for your network camera's video stream is usually the only challenge in connecting this component. Most, if not all, network cameras have an internal web server, allowing viewers to use web browsers to view their video stream. If you go to that webpage, and look at the HTML source of the page, you should be able to find the URL of the MJPEG or JPEG still stream.

Some examples:

Axis 2100 (MJPEG): `http://ip.address.here/axis-cgi/mjpg/video.cgi?resolution=640x480`

Panasonic BL-C10A (MJPEG): `http://ip.address.here/nphMotionJpeg?Resolution=640x480&Quality=Standard`

StarDot Netcam (JPEG stills): `http://ip.address.here/netcam.jpg`

© 2011 Inductive Automation

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the component.

Scripting name	`background`
Data type	Color

Show Stats — If true, fps and Kbps statistical information will be overlaid on the video.

Scripting name	`showStats`
Data type	boolean

Behavior

Video Mode — Choose what type of video stream the URL points to.

Scripting name	`mode`
Data type	int
Values	`0` MJPEG Stream
	`1` JPEG Stills

Refresh Rate — The rate (in ms) to poll the image if mode is 'JPEG Stills'

Scripting name	`refreshRate`
Data type	int

Use Authentication? — If true, the URL connection will try to authenticate using the given username and password.

Scripting name	`useAuthentication`
Data type	boolean

Username — The username to authenticate with.

Scripting name	`username`
Data type	String

Password — The password to authenticate with.

Scripting name	`password`
Data type	String

URL — The HTTP URL of the video stream to display

Scripting name	`url`
Data type	String

User-Agent — If non-empty, the HTTP User-Agent to spoof.

Scripting name	`userAgent`
Data type	String
Flags	expert

Scale Video — Scale the video to the size of the viewer component. Warning: CPU-intensive.

Scripting name	`scaleVideo`
Data type	boolean

© 2011 Inductive Automation

Flags expert

Scale Mode

The scaling performance hint to use.

Scripting name `scaleMode`
Data type int
Flags expert
Values
- 1 Default
- 2 Fast
- 4 Smooth
- 16 Area Averaging
- 8 Replicate

Connection Retries

The number of times to attempt to connect to the stream.

Scripting name `connectRetries`
Data type int

Retry Delay

The delay (in ms) to wait between connection attempts

Scripting name `retryDelay`
Data type int

Common

Name

The name of this component.

Scripting name `name`
Data type String
Flags bindable

Visible

If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor

The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values
- 0 Default
- 1 Crosshair
- 2 Text
- 3 Wait
- 12 Hand
- 13 Move
- 4 SW Resize
- 5 SE Resize
- 6 NW Resize
- 7 NE Resize
- 8 N Resize
- 9 S Resize
- 10 W Resize
- 11 E Resize

Scripting

© 2011 Inductive Automation

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

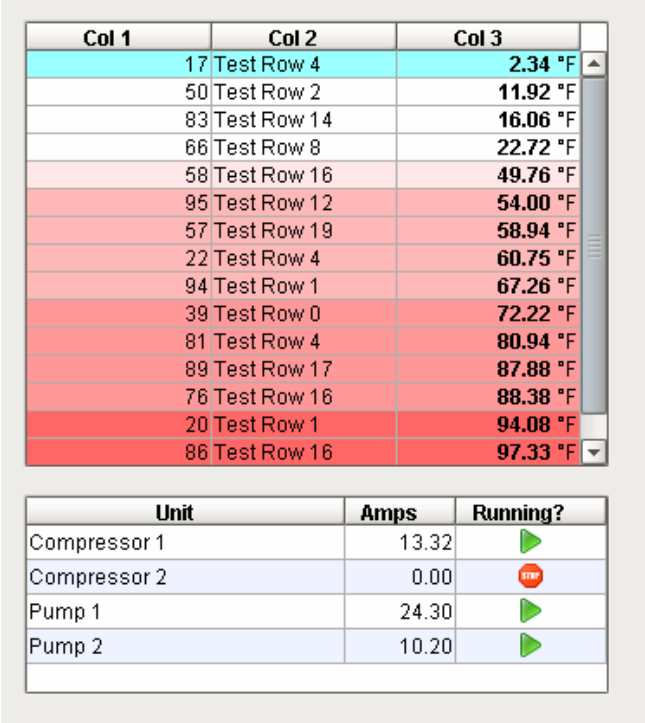
mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.4 Tables

7.4.1 Table



Col 1	Col 2	Col 3
17	Test Row 4	2.34 °F
50	Test Row 2	11.92 °F
83	Test Row 14	16.06 °F
66	Test Row 8	22.72 °F
58	Test Row 16	49.76 °F
95	Test Row 12	54.00 °F
57	Test Row 19	58.94 °F
22	Test Row 4	60.75 °F
94	Test Row 1	67.26 °F
39	Test Row 0	72.22 °F
81	Test Row 4	80.94 °F
89	Test Row 17	87.88 °F
76	Test Row 16	88.38 °F
20	Test Row 1	94.08 °F
86	Test Row 16	97.33 °F

Unit	Amps	Running?
Compressor 1	13.32	
Compressor 2	0.00	
Pump 1	24.30	
Pump 2	10.20	

Two tables showing a variety of display options

Description

The Table component is very powerful and easy to configure. It is very flexible, allowing you to easily display your tabular data in a variety of ways. Important features include:

Column Sorting. Your users can easily sort the data by clicking on the column headers. The sorting is a 3-mode sort: Ascending, Descending, and "Natural", which uses the default order of the data.

Mapped Row Coloring. Map the background color of each row to a particular column. This allows you to give powerful visual indication of different types of rows in you tables, such as differentiating between alarm states.

Column Translation. Allow the table component to handle all code mapping, such as mapping 0 to "Off" and 1 to "On". No fancy SQL knowledge required.

Images. Map values to images, allowing intuitive visual cues.

Progress Bar Indication. Display numeric data as progress bars inside cells, providing fast visual reference for bounded amounts.

© 2011 Inductive Automation

Number and Date formatting. Format numbers and dates to your exact specification.
Column Hiding. Hide columns from view that contain identifying data used by the row coloring or by other components.
Printing. Print tables directly to multi-paged printouts.
Editing. Columns can be made editable. Changes will be reflected in the underlying dataset, at which point they can be mapped back to a database.

Basic Usage

The basic usage of the Table is to use a SQL Query binding on its *Data* property to let the table display data from a database. Often this query will by dynamic or indirect. See the Property Binding section for more information.

Binding to Selected Data

It is common to want to bind other components to values in the selected row of the table. In order to do this safely, you need to write an expression binding that protects against the case when nothing is selected or there are now rows. An expression like this would bind a label to the selected row's value for a column named "ProductCode":

```
if({Root Container.MyTable.selectedRow} = -1,
    "n/a", // this is the fail case
    {Root Container.MyTable.data}[{Root Container.MyTable.selectedRow}, "ProductCode"])
```

If you're binding to an integer, date, or other non-String type value thats inside a dateset, you'll need to *cast* the value to the correct type to make the expression parser happy. This binding would cast the selected "Quantity" column to an integer:

```
if({Root Container.MyTable.selectedRow} = -1,
    -1, // this is the fail case
    toInt({Root Container.MyTable.data}[{Root Container.MyTable.selectedRow}, "Quantity"]))
```

Changing the Column Widths

To change a table's column's widths, simply switch into preview mode and use your mouse to resize the columns, and then switch back to design mode. Simple!

Editable Table

By setting any column to *editable* in the Table's customizer, the user will be able to double-click in the cell and edit the data. You can the respond to the resulting `cellEdited` event with an event handler and persist the data. See the Event Types section for more information.

Exporting to HTML

You can export the table to an HTML file that retain's the table's formatting. To do this, use a script like this: (more about the table's exportHTML function is here.)

```
table = event.source.parent.getComponent("Table") # Get a reference to the table
table.exportHTML("MyTable.html", "My Table Header", 500) # Prompt user to save the exported fil
```

Exporting to CSV

You can export the table's raw data to a CSV file. To do this, use a script like this: (more about the fpmi.db.exportCSV function is here.)

```
table = event.source.parent.getComponent("Table") # Get a reference to the table
system.dataset.exportCSV("mydata.csv", 1, table.data)
```

Printing

Printing a table is a snap! Simply use the table's built in `print` function like this:
table = event.source.parent.getComponent("Table") # Get a reference to the table
table.print()

© 2011 Inductive Automation

See also:
SQL Query Binding
Expression Binding
Event Types - cellEdited
system.dataset.exportCSV
system.dataset.dataSetToExcel
system.dataset.dataSetToHTML
system.print.createPrintJob

Properties

Appearance

Font — Font of text of this component
Scripting name: `font`
Data type: Font

Foreground Color — The foreground color of the component.
Scripting name: `foreground`
Data type: Color

Background Color — The background color of the component.
Scripting name: `background`
Data type: Color

Header Font — Font of the table's header text
Scripting name: `headerFont`
Data type: Font

Header Foreground Color — The foreground color of the table's header.
Scripting name: `headerForeground`
Data type: Color

Header Visible — Whether or not the table header is visible.
Scripting name: `headerVisible`
Data type: boolean

Row Height — The height of each row, in pixels
Scripting name: `rowHeight`
Data type: int

Background Mode — This mode determines the color that this table's cell's backgrounds will be.
Scripting name: `backgroundColorMode`
Data type: int
Values:
1 Constant
2 Alternating
3 Mapped

Odd Row Background — The color which odd rows will be colored if background mode is 'Alternating'
Scripting name: `oddBackground`
Data type: Color

Selection Background — The background color of a selected cell.
Scripting name: `selectionBackground`
Data type: Color

© 2011 Inductive Automation

Flags expert

Selection Foreground — The foreground color of a selected cell.

Scripting name `selectionForeground`
Data type Color
Flags expert

Show Horizontal Grid Lines?

Scripting name `showHorizontalLines`
Data type boolean
Flags expert

Show Vertical Grid Lines?

Scripting name `showVerticalLines`
Data type boolean
Flags expert

Grid Line Color — The color used to draw grid lines.

Scripting name `gridColor`
Data type Color
Flags expert

Behavior

Selection Mode — This mode determines if only one row/cell/column can be selected at once, or single or multiple intervals

Scripting name `selectionMode`
Data type int
Values
- **0** Single
- **1** Single Interval
- **2** Multiple Interval

Row Selection Allowed — This flag is used in conjunction with the Column Selection Allowed flag to determine whether not whole-rows, whole-columns, or both (single-cells) are selectable.

Scripting name `rowSelectionAllowed`
Data type boolean

Column Selection Allowed — This flag is used in conjunction with the Row Selection Allowed flag to determine whether not whole-rows, whole-columns, or both (single-cells) are selectable.

Scripting name `columnSelectionAllowed`
Data type boolean

Resizing Allowed — Whether or not the user is allowed to resize table headers or not.

Scripting name `resizingAllowed`
Data type boolean

Auto-Resize Mode — Determines how the table resizes the columns

Scripting name `autoResizeMode`
Data type int
Values
- **4** All Columns
- **3** Last Column
- **1** Next Column
- **0** Off
- **2** Subsequent Columns

Initially Selected Row — The index of the row that should be selected by default when this table's data is filled in. Note that you must save the table with no selection in

© 2011 Inductive Automation

order for this to work.

Scripting name	`initialRowSelection`
Data type	int
Flags	expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name	`opaque`
Data type	boolean

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default **1** Crosshair **2** Text **3** Wait **12** Hand **13** Move **4** SW Resize **5** SE Resize **6** NW Resize **7** NE Resize **8** N Resize **9** S Resize **10** W Resize **11** E Resize

Data

Data — The data for this table

Scripting name	`data`
Data type	Dataset
Flags	bindable

© 2011 Inductive Automation

Column Attributes Data — The dataset describing the column attributes.

Scripting name	`columnAttributesData`
Data type	Dataset
Flags	expert

Selected Column — The index of the first selected column, or -1 if none.

Scripting name	`selectedColumn`
Data type	int
Flags	bindable | expert

Selected Row — The index of the first selected row, or -1 if none.

Scripting name	`selectedRow`
Data type	int
Flags	bindable | expert

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Uncategorized

TestData — Toggle this property to fill in the table's data with random data.

Scripting name	`test`
Data type	boolean
Flags	expert

Properties Loading — The number of properties currently being loaded

Scripting name	`propertiesLoading`
Data type	int
Flags	bindable | read-only

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- cell
- focus
- propertyChange
- key

Scripting Functions

`addRow(`newRow**`)`** — Adds a new row to the end of the table's dataset.

Parameters	PySequence `newRow` - A sequence containing the values for the new row. The length of the sequence must match the number of columns in the table, and each value must be coercible into the correct datatype of the corresponding column.
Returns	nothing

`deleteRow(`rowIndex**`)`** — Deletes a row from the table's dataset.

Parameters	int `rowIndex` - The index of the row to delete.
Returns	nothing

`exportCSV(`filename, shMissing description.

© 2011 Inductive Automation

Parameters String `filename`
boolean `showHeaders`
Returns String

exportHTML(filename, tPrompts the user to save the table's data as an html file.

Parameters String `filename` - A suggested filename for the user. For example: "table_data.html"
String `title` - The title for the HTML page.
int `width` - The width (in pixels) for the "table" element in the resulting html page.
Returns String

getDataAsHTML(title, \Creates an HTML page as a string in memory. This can then be written to a file, a database, emailed, etc.

Parameters String `title` - The title for the HTML page.
int `width` - The width (in pixels) for the "table" element in the resulting html page.
Returns String - A string containing an HTML-formatted version of the table's data.

getRowsInViewOrdeReturns a list of ints that represent the underlying dataset's rows as they appear in the current sort order that the user is viewing.

Parameters none
Returns int[]

getSelectedColumnReturns the index of the currently selected column, or -1 if none is selected.

Parameters none
Returns int

getSelectedColumnReturns the number of columns that are currently selected.

Parameters none
Returns int

getSelectedColumnReturns a list of ints representing the currently selected columns.

Parameters none
Returns int[]

getSelectedRow() Returns the index of the currently selected row, or -1 if none is selected.

Parameters none
Returns int

getSelectedRowCouReturns the number of rows that are currently selected.

Parameters none
Returns int

getSelectedRows() Returns a list of ints that represent the currently selected rows.

Parameters none
Returns int[]

isCellSelected(rowTests whether the cell at the given row and column is currently selected or not.

Parameters int `row`
int `column`
Returns boolean - 1 or 0 meaning selected or not selected, respectively.

isColumnSelected(Tests whether the given column is currently selected or not.

© 2011 Inductive Automation

Parameters int column
Returns boolean

isRowSelected(row**)** Tests whether the given row is currently selected or not.

Parameters int row
Returns boolean

print(??, ??**)** This specialized print function will paginate the table onto multiple pages.

Parameters PyObject[] ??
String[] ??
Returns boolean

setColumnLabel(col Used to set a column's header label to a new string at runtime.

Parameters int column
String label
Returns nothing

setColumnSelectic Sets the given range of columns to be selected. If index0==index1, it will select a single column.

Parameters int index0
int index1
Returns boolean

setColumnWidth(col Used to set a column's width at runtime.

Parameters int column
int width
Returns nothing

setRowSelectionIn Sets the given range of rows to be selected. If index0==index1, it will select a single row.

Parameters int index0
int index1
Returns boolean

setSelectedColumn Sets the given column to be the selected column.

Parameters int ??
Returns nothing

setSelectedRow(??, Sets the given row to be the selected row.

Parameters int ??
Returns nothing

setValue(row, column, v Sets the value in the specified cell, altering the table's Data property. Will fire a propertyChange event for the "data" property, as well as a cellEdited event.

Parameters int row - The index of the row to set the value at.
int column - The index or name of the column to set a value at.
PyObject value - The new value to use at the given row / column location.
Returns nothing

sortByColumn(colum Instructs the table to sort the data by the named column.

Parameters String columnName
boolean asc - 1 means ascending, 0 means descending. (default = 1) [optional]
Returns nothing

© 2011 Inductive Automation

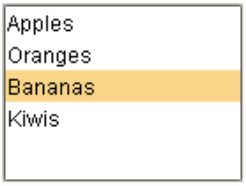
sortOriginal() Instructs the table to clear any custom sort columns and display the data as it is sorted in the underlying dataset.

Parameters none
Returns nothing

updateRow(rowIndex, chUpdates an entire row of the table's dataset.

Parameters int `rowIndex` - The index of the row to update.
PyDictionary `changes` - A sequence containing the updated values for the row. The length of the sequence must match the number of columns in the table, and each value must be coercible into the correct datatype of the corresponding column.
Returns nothing

7.4.2 List



A basic List

Description

The List component displays a list of options, allowing freeform selection of the items. It is powered by a Dataset, from which it displays the first column.

Properties

Appearance

Font — Font of text of this component
Scripting name `font`
Data type Font

Foreground Color — The foreground color of the component.
Scripting name `foreground`
Data type Color

Background Color — The background color of the component.
Scripting name `background`
Data type Color

Selected Foreground — The color of the foreground for the selected cell(s).
Scripting name `selectedForeground`
Data type Color

Selected Background — The color of the background for the selected cell(s).
Scripting name `selectedBackground`
Data type Color

Selected Focus Border — The border for the selected, focused cell.
Scripting name `selectedFocusBorder`
Data type Border

Styles — Contains the component's styles
Scripting name `styles`

© 2011 Inductive Automation

Data type	Dataset
Flags	bindable | expert

Behavior

Selection Mode

This mode determines if only one cell can be selected at once, or single or multiple intervals

Scripting name	`selectionMode`
Data type	int
Values	0 Single 1 Single Interval 2 Multiple Interval

Common

Name

The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled

If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible

If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Opaque

If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name	`opaque`
Data type	boolean

Cursor

The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	0 Default 1 Crosshair 2 Text 3 Wait 12 Hand 13 Move 4 SW Resize 5 SE Resize 6 NW Resize 7 NE Resize 8 N Resize 9 S Resize 10 W Resize 11 E Resize

Data

© 2011 Inductive Automation

Data — The data for the list. If multiple columns exist, the first will be used.

Scripting name	`data`
Data type	Dataset
Flags	bindable

Selected Index — The index of the selected cell, or -1 if none.

Scripting name	`selectedIndex`
Data type	int
Flags	bindable | expert

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
focus
propertyChange
key

Scripting Functions

`addSelectionInter` Adds the options at indexes start through end (inclusive) to the selected options.

Parameters	int `start` – The first index (stating at 0) to add to the selection. int `end` – The last index (stating at 0) to add to the selection.
Returns	nothing

`clearSelection()` Clears the current selection, making nothing selected.

Parameters	none
Returns	nothing

`getSelectedIndice` Returns a list of the selected indices in increasing order. Returns an empty list if nothing is selected.

Parameters	none
Returns	int[]

`getSelectedValue(` Returns the currently selected value, or None if the selection is empty

Parameters	none
Returns	Object

`getSelectedValues` Returns a list of the currently selected values. Returns an empty list if the selection is empty.

Parameters	none
Returns	Object[]

`isSelectedIndex(``ir` Checks whether or not the given index is currently selected.

Parameters	int `index`
Returns	boolean

© 2011 Inductive Automation

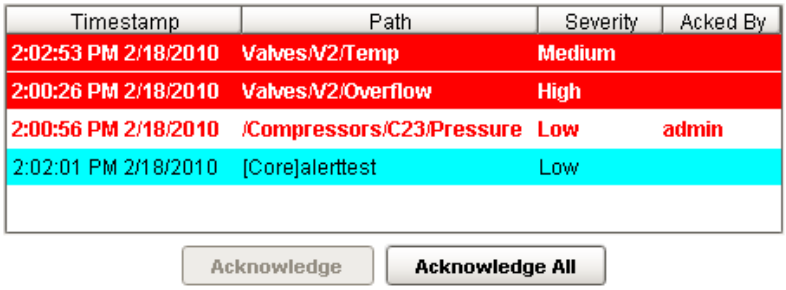
isSelectionEmpty (Checks to see if anything is selected in the list or not.

Parameters	none
Returns	boolean

setSelectedValue (Sets the currently selected value to the argument, if found in the list.

Parameters	Object `value`
Returns	nothing

7.4.3 Alert Summary Table



Description

The alert summary table provides an easy way to display current and unacknowledged alerts, and to provide acknowledgement functionality for your alerts.

Properties

Alert Styles

Active and Unacked Foreground 1

Scripting name	`activeAndUnackedForeground1`
Data type	Color

Active and Unacked Background 1

Scripting name	`activeAndUnackedBackground1`
Data type	Color

Active and Unacked Foreground 2

Scripting name	`activeAndUnackedForeground2`
Data type	Color

Active and Unacked Background 2

Scripting name	`activeAndUnackedBackground2`
Data type	Color

Active and Unacked Blink?

Scripting name	`activeAndUnackedBlink`
Data type	boolean

Active and Unacked Font

Scripting name	`activeAndUnackedFont`
Data type	Font

Active and Acked Foreground 1

Scripting name	`activeAndAckedForeground1`
Data type	Color

© 2011 Inductive Automation

Active and Acked Background 1
Scripting name activeAndAckedBackground1
Data type Color

Active and Acked Foreground 2
Scripting name activeAndAckedForeground2
Data type Color

Active and Acked Background 2
Scripting name activeAndAckedBackground2
Data type Color

Active and Acked Blink?
Scripting name activeAndAckedBlink
Data type boolean

Active and Acked Font
Scripting name activeAndAckedFont
Data type Font

Clear and Unacked Foreground 1
Scripting name clearAndUnackedForeground1
Data type Color

Clear and Unacked Background 1
Scripting name clearAndUnackedBackground1
Data type Color

Clear and Unacked Foreground 2
Scripting name clearAndUnackedForeground2
Data type Color

Clear and Unacked Background 2
Scripting name clearAndUnackedBackground2
Data type Color

Clear and Unacked Blink?
Scripting name clearAndUnackedBlink
Data type boolean

Clear and Unacked Font
Scripting name clearAndUnackedFont
Data type Font

Clear and Acked Foreground 1
Scripting name clearAndAckedForeground1
Data type Color

Clear and Acked Background 1
Scripting name clearAndAckedBackground1
Data type Color

Clear and Acked Foreground 2
Scripting name clearAndAckedForeground2
Data type Color

Clear and Acked Background 2

© 2011 Inductive Automation

Scripting name `clearAndAckedBackground2`
Data type Color

Clear and Acked Blink?

Scripting name `clearAndAckedBlink`
Data type boolean

Clear and Acked Font

Scripting name `clearAndAckedFont`
Data type Font

Appearance

Header Visible? Should the alert table have a header row?

Scripting name `headerVisible`
Data type boolean

Table Background The background color for the empty space in the table

Scripting name `tableBackground`
Data type Color

Table Border The border around the table itself, not including the controls.

Scripting name `scrollPaneBorder`
Data type Border

Selection Color The color of the selection border

Scripting name `selectionColor`
Data type Color

Selection Thickness The size of the selection border

Scripting name `selectionThickness`
Data type int

Row Height The height, in pixels, for each row of the table.

Scripting name `rowHeight`
Data type int

Blink On-Time The amount of time (in millis) to display the "blink-on" state.

Scripting name `blinkOnTime`
Data type int

Blink Off-Time The amount of time (in millis) to display the "blink-off" state.

Scripting name `blinkOffTime`
Data type int

Date Format A date format pattern used to format dates in the table.

Scripting name `dateFormat`
Data type String

Number Format A number format pattern used to format alert values.

Scripting name `numberFormat`
Data type String

Ack Buttons Location The location of the acknowledgement button panel

Scripting name `ackButtonLocation`
Data type int
Values
- `1` North
- `3` East
- `5` South
- `7` West

© 2011 Inductive Automation

	-1 Hidden

Ack Button Text — The text for the acknowledgement button.

Scripting name	ackText
Data type	String

Ack All Button Text — The text for the acknowledge-all button.

Scripting name	ackAllText
Data type	String

Ack Button Font — The font for the acknowledgement buttons

Scripting name	ackButtonFont
Data type	Font

Notes Area Location — The location of the notes display area

Scripting name	notesAreaLocation
Data type	int
Values	1 North
	3 East
	5 South
	7 West
	-1 Hidden

Notes Area Size — The size of the notes area, in pixels.

Scripting name	notesAreaSize
Data type	int

Notes Area Border — The border surrounding the notes area.

Scripting name	notesAreaBorder
Data type	Border

Notes Area Font — The font for the notes area.

Scripting name	notesAreaFont
Data type	Font

Behavior

Refresh Rate — The rate at which this table will poll for new alerts.

Scripting name	refreshRate
Data type	long

Flatten Alerts — If true, only one alert state will be shown for any alert. The most recent and severe alert state will be chosen.

Scripting name	flatten
Data type	boolean
Flags	expert

Auto-Resize Mode — Determines how the table resizes the columns

Scripting name	autoResizeMode
Data type	int
Values	4 All Columns
	3 Last Column
	1 Next Column
	0 Off
	2 Subsequent Columns

Columns

Column Timestamp Visible?

Scripting name	showTimestamp

© 2011 Inductive Automation

Data type boolean

Column Value Visible?

Scripting name showValue
Data type boolean

Column System Visible?

Scripting name showSystem
Data type boolean

Column ItemPath Visible?

Scripting name showItemPath
Data type boolean

Column Path Visible?

Scripting name showPath
Data type boolean

Column State Visible?

Scripting name showState
Data type boolean

Column Severity Visible?

Scripting name showSeverity
Data type boolean

Column Cleared Visible?

Scripting name showCleared
Data type boolean

Column Clear Value Visible?

Scripting name showClearValue
Data type boolean

Column Acked Visible?

Scripting name showAcked
Data type boolean

Column Acked By Visible?

Scripting name showAckedBy
Data type boolean

Column Timestamp Width

Scripting name columnTimestampWidth
Data type int

Column Value Width

Scripting name columnValueWidth
Data type int

Column System Width

Scripting name columnSystemWidth
Data type int

Column ItemPath Width

Scripting name columnItemPathWidth
Data type int

© 2011 Inductive Automation

Column Path Width

Scripting name `columnPathWidth`
Data type int

Column State Width

Scripting name `columnStateWidth`
Data type int

Column Severity Width

Scripting name `columnSeverityWidth`
Data type int

Column Cleared Width

Scripting name `columnClearedWidth`
Data type int

Column Clear Value Width

Scripting name `columnClearValueWidth`
Data type int

Column Acked Width

Scripting name `columnAckedWidth`
Data type int

Column Acked By Width

Scripting name `columnAckedByWidth`
Data type int

Column Timestamp Header

Scripting name `columnTimestampText`
Data type String

Column Value Header

Scripting name `columnValueText`
Data type String

Column System Header

Scripting name `columnSystemText`
Data type String

Column ItemPath Header

Scripting name `columnItemPathText`
Data type String

Column Path Header

Scripting name `columnPathText`
Data type String

Column State Header

Scripting name `columnStateText`
Data type String

Column Severity Header

Scripting name `columnSeverityText`
Data type String

Column Cleared Header

© 2011 Inductive Automation

Scripting name columnClearedText
Data type String

Column Clear Value Header

Scripting name columnClearValueText
Data type String

Column Acked Header

Scripting name columnAckedText
Data type String

Column Acked By Header

Scripting name columnAckedByText
Data type String

Column Timestamp Position

Scripting name columnTimestampPosition
Data type int
Flags expert

Column Value Position

Scripting name columnValuePosition
Data type int
Flags expert

Column System Position

Scripting name columnSystemPosition
Data type int
Flags expert

Column ItemPath Position

Scripting name columnItemPathPosition
Data type int
Flags expert

Column Path Position

Scripting name columnPathPosition
Data type int
Flags expert

Column State Position

Scripting name columnStatePosition
Data type int
Flags expert

Column Severity Position

Scripting name columnSeverityPosition
Data type int
Flags expert

Column Cleared Position

Scripting name columnClearedPosition
Data type int
Flags expert

Column Clear Value Position

Scripting name columnClearValuePosition
Data type int
Flags expert

© 2011 Inductive Automation

Column Acked Position

Scripting name	`columnAckedPosition`
Data type	int
Flags	expert

Column Acked By Position

Scripting name	`columnAckedByPosition`
Data type	int
Flags	expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Data

Selected Row — The currently selected row

Scripting name	`selectedRow`
Data type	int
Flags	bindable | expert

Alerts — A dataset holding the alerts that the table is currently displaying. Read-only.

Scripting name	`alerts`
Data type	Dataset
Flags	bindable | expert

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Filters

System Filter — Filter alerts to a specific system. Use * and ? to match any characters or one character, respectively.

Scripting name	`systemFilter`
Data type	String

Item Path Filter — Filter alerts by item path. Use * and ? to match any characters or one character, respectively.

Scripting name	`itemPathFilter`
Data type	String

© 2011 Inductive Automation

Flags expert

Path Filter
Filter alerts by display path, or item path if no display path is set. Use * and ? to match any characters or one character, respectively.
Scripting name `pathFilter`
Data type String

Min Severity
The minimum severity to display
Scripting name `severityFilter`
Data type int
Values
0 Low
1 Medium Low
2 Medium
3 Medium High
4 High

Show Active and Unacked
If true, alerts that are active and unacknowledged will be displayed.
Scripting name `activeAndUnacked`
Data type boolean

Show Active and Acked
If true, alerts that are active and acknowledged will be displayed.
Scripting name `activeAndAcked`
Data type boolean

Show Clear and Unacked
If true, alerts that are cleared and unacknowledged will be displayed.
Scripting name `clearAndUnacked`
Data type boolean

Show Clear and Acked
If true, alerts that are cleared and acknowledged will be displayed.
Scripting name `clearAndAcked`
Data type boolean

Sort Order

Sort by Active
Sort priority for sorting by the alert's active state.
Scripting name `sortByActive`
Data type int

Sort by Acked
Sort priority for sorting by the alert's acknowledgement state.
Scripting name `sortByAcked`
Data type int

Sort by Active Time
Sort priority for sorting by the alert's active timestamp.
Scripting name `sortByActiveTime`
Data type int

Sort by Severity
Sort priority for sorting by the alert's severity.
Scripting name `sortBySeverity`
Data type int

Sort by System
Sort priority for sorting by the alert's originating system.
Scripting name `sortBySystem`
Data type int

Sort by Path
Sort priority for sorting by the alert's display path.
Scripting name `sortByPath`
Data type int

Sort by State Name
Sort priority for sorting by the alert's state name.

© 2011 Inductive Automation

	Scripting name `sortByStateName` Data type int
Sort by Clear Time	Sort priority for sorting by the alert's cleared timestamp. Scripting name `sortByClearTime` Data type int
Sort by Acked Time	Sort priority for sorting by the alert's acknowledgement timestamp. Scripting name `sortByAckedTime` Data type int

Scripting

Events

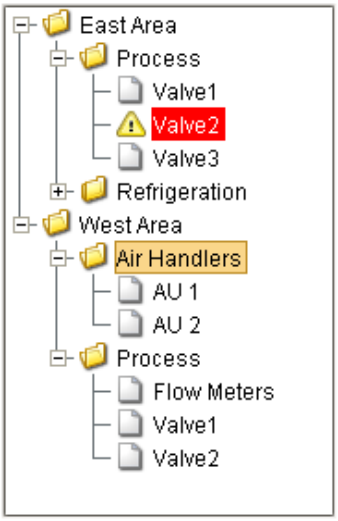
The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- focus
- propertyChange
- key

Scripting Functions

This component has no special scripting functions.

7.4.4 Tree View



Trees are useful for navigating hierarchies.

Description

The Tree View component can display any tree hierarchy. It is configured by filling in a dataset. Each row in the dataset will become a node in the tree. Each node has a path, for example, "`West Area/Process/Valve1`" that determines its location in the tree. The *Separation Character* property (by default it is forward-slash), dictates how the paths are broken up. Any missing folder nodes needed by a leaf node are created implicitly.

The other columns in the dataset besides "Path" are used to configure the look for the node, both when it is selected and when it is not. Columns with the following names (case-insensitive) in the

© 2011 Inductive Automation

dataset will be recognized:

Path - the path determines the node's location. Broken up into a list by splitting on the separation character.
Text - the text of the node while not selected.
Icon - a path to an icon for the node. Use the value: "`default`" to use the tree automatic folder/leaf icons.
Background - a string column that will be coerced into a color for the unselected background. e.g. "`white`" or "`(255,255,255)`" Use an empty string to use the default color.
Foreground - a string representation of the unselected foreground color
Tooltip - if not empty, will be used as the tooltip for the node.
Border - a string that will be coerced into a Border for the node while unselected. May be empty.
SelectedText - the text of the node while selected.
SelectedIcon - a path to an icon for the node while selected. Use the value: "`default`" to use the tree automatic folder/leaf icons.
SelectedBackground - a string representation of the selected foreground color
SelectedForeground - a string representation of the selected foreground color
SelectedTooltip - if not empty, will be used as the tooltip for the node while selected.
SelectedBorder - a string that will be coerced into a Border for the node while selected. May be empty.

The Selected Item property will be updated as the user selects different nodes in the tree. It represents the index in the Items dataset at which the node is defined. If the selected node was implicitly created, the Selected Item will be -1. You can use this index to get the path and name of the selected node with an expression binding like this:

```
if ({Root Container.Tree View.selectedItem}<0,"n/a",
{Root Container.Tree View.data}[{Root Container.Tree View.selectedItem},"text"])
```

Properties

Appearance

Font — Font of text of this component
Scripting name: `font`
Data type: Font

Background Color — The background color of the component.
Scripting name: `background`
Data type: Color

Row Height — The height of each row in the tree
Scripting name: `rowHeight`
Data type: int

Show Root Handles — Whether or not to show handles next to parent nodes
Scripting name: `showRootHandles`
Data type: boolean

Default Node Background — The default background of a node if no background is set
Scripting name: `defaultBackground`
Data type: Color
Flags: expert

Default Node Foreground — The default foreground of a node if no foreground is set
Scripting name: `defaultForeground`
Data type: Color

© 2011 Inductive Automation

Flags	expert

Default Node Border

The default border of a node if no border is set

Scripting name	`defaultBorder`
Data type	Border
Flags	expert

Default Node Selected Background

The default selected background of a node if no background is set

Scripting name	`defaultSelectedBackground`
Data type	Color
Flags	expert

Default Node Selected Foreground

The default selected foreground of a node if no foreground is set

Scripting name	`defaultSelectedForeground`
Data type	Color
Flags	expert

Default Node Selected Border

The default selected border of a node if no border is set

Scripting name	`defaultSelectedBorder`
Data type	Border
Flags	expert

Default Leaf Icon

The default leaf icon if no icon is set

Scripting name	`defaultLeafIconPath`
Data type	String
Flags	expert

Default Open Icon

The default open icon if no icon is set

Scripting name	`defaultOpenIconPath`
Data type	String
Flags	expert

Default Closed Icon

The default closed icon if no icon is set

Scripting name	`defaultClosedIconPath`
Data type	String
Flags	expert

Line Style

The tree's line style

Scripting name	`lineStyle`
Data type	int
Flags	expert
Values	0 Angled 2 None

Behavior

Separation Character

The separation character for the path

Scripting name	`separationCharacter`
Data type	String

Auto Sort

Whether or not to automatically sort the tree

Scripting name	`autoSort`
Data type	boolean

Auto Expand

Whether or not to automatically expand all nodes in the tree

Scripting name	`autoExpand`
Data type	boolean

Selection Mode

What kind of selection regions does the tree allow.

© 2011 Inductive Automation

Scripting name	`selectionMode`
Data type	int
Values	**1** Single Selection **2** Multiple - Contiguous **4** Multiple - Discontiguous

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Data

Items — Contains the items of the tree view

Scripting name	`data`
Data type	Dataset
Flags	bindable

Selected Item — The index of the currently selected item, or -1 if no selection.

Scripting name	`selectedItem`
Data type	int
Flags	bindable

Selected Path — The path of the currently selected item, or "" if no selection.

Scripting name	`selectedPath`
Data type	String
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion

© 2011 Inductive Automation

propertyChange

Scripting Functions

`clearSelection()` Clears the current selection.

Parameters	none
Returns	nothing

`collapseAll()` Collapses all nodes in the tree.

Parameters	none
Returns	nothing

`expandAll()` Expands all nodes in the tree.

Parameters	none
Returns	nothing

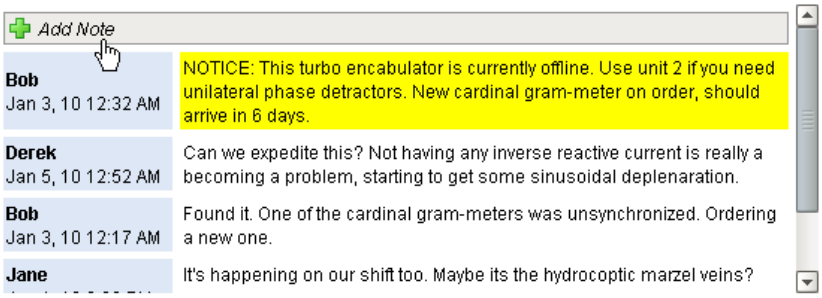
`getSelectedItems()` Returns a list of the selected item's indexes. These are the row indexes that the selected tree nodes were found in the underlying dataset. Implicitly created folder nodes that have no index will not be included.

Parameters	none
Returns	int[]

`getSelectedPaths()` Returns a list of the selected item's paths. A path to an item is the path to its parent plus its normal (non-selected) text.

Parameters	none
Returns	String[]

7.4.5 Comments Panel


Add Note
Bob
Jan 3, 10 12:32 AM
NOTICE: This turbo encabulator is currently offline. Use unit 2 if you need unilateral phase detractors. New cardinal gram-meter on order, should arrive in 6 days.
Derek
Jan 5, 10 12:52 AM
Can we expedite this? Not having any inverse reactive current is really a becoming a problem, starting to get some sinusoidal deplenaration.
Bob
Jan 3, 10 12:17 AM
Found it. One of the cardinal gram-meters was unsynchronized. Ordering a new one.
Jane
It's happening on our shift too. Maybe its the hydrocoptic marzel veins?

The database-powered Comments Panel helps operator collaboration.

Description

The comments panel is used to power a blog-style comments system within your project. This can be useful for ad-hoc collaboration and communication between shifts, remote users, etc. This component is driven by a dataset that should be bound to a SQL query. Unlike most components, this component has built-in functionality to alter an external database. This is how the Add Note functionality works. You have the opportunity to alter the queries that the components uses by changing their properties.

The schema that typically drives this component involves up to two tables. One table (by default: *Notes*) stores all of the notes across the board. The second table (by default, *ItemNotes*) is used to associate notes with other things. This allows you to have different sets of notes for different screens/ objects. Typically you'd bind the data to a query that joined these tables together restricting the second identifier in the ItemNotes table to the value appropriate for the window you're on. You'll also need to alter *Insert Query 2*'s "`YOURID`" placeholder so that new notes get put in the right spot. You

can opt out of this two-table system by simply clearing out *Insert Query 2*.

Users can be given the choice to remove their own comments, and comments can have files attached. To allow attachments, make sure you have a BLOB field in your notes table.

This component expects that its dataset is populated with the following columns. The names do not need to be exact, but the data type in your notes table must match.

ID - an integer that should be the primary key for the notes table. Used for deleting and looking up attachments.
Username - the user who added the note. Must be a string/varchar.
Timestamp - when the note was added. Must be a Date or DateTime data type.
NoteText - The text of the note itself. Must be a string/varchar.
AttachmentFilename - filename for a file attached to the note. Must be a string/varchar.
Stick - 0 or 1 indicating whether or note the note is "sticky", which means it gets highlighted and put at the top. Must be a boolean or integer.

A short explanation for each of the queries and what is passed into them automatically. Note that the column names here do not need to match the ones in the Data property.

Insert Query 1: INSERT INTO Notes (Note, WhoID, TStamp, Attachment, Filename, Sticky) VALUES (?, (SELECT Id FROM Users WHERE Username='%s'), CURRENT_TIMESTAMP, ?, ?, ?)

This query will insert into your note table using the runPrepStmtGetKey() function and will be given four variables in the following order: note text, attachment blob, attachment filename, and sticky value. Also, it will pass in one string denoted by the %s. This is the name of the user that entered the note and does not need to be placed in any specific spot. *If you WhoID field is a string, you can replace (SELECT Id FROM Users WHERE Username='%s') with '%s' to pass the username in directly.*

Insert Query 2: INSERT INTO ItemNotes (AccountId, NoteId) VALUES (YOURID, %d)

This query is optional and will insert the note id from Insert Query 1 into a mapping table of your choice. You must replace YOURID with something meaningful for your mapping table. This is most commonly done by binding this query to an expression. The reason for this second query is to have a mapping table to be joined to the note table to filter out which notes belong to a particular Comment Panel component.

Delete Query: DELETE FROM Notes WHERE Id=%d

This query will use the note id from the component to delete the selected note.

Unstick Query: UPDATE Notes SET Sticky=0 WHERE Id=%d

This query will use the note id from the component to set the sticky value to 0.

Download Attachment Query: SELECT Attachment FROM Notes WHERE Id=%d

This query will use the note id from the component to download the attachment blob from the database.

Sample queries for the Data property binding:
Note that the data types in the database must be correct and the columns must be in this order

Assuming WhoID is a string that contains the username:

```
SELECT ID, WhoID as 'Username', TStamp as 'Timestamp', Note as 'NoteText', Filename as 'Attachm
FROM notes
ORDER BY TStamp DESC
```

If WhoID is a foreign key linked to the Users Table:

```
SELECT n.ID, u.Username, n.TStamp, n.Note, n.Filename, n.Sticky
FROM notes n
INNER JOIN users u ON u.ID = n.WhoID
ORDER BY TStamp DESC
```

© 2011 Inductive Automation

If WhoID is a string and your ItemNotes table links AccountId to NoteId

```
SELECT n.ID, n.WhoID, n.TStamp, n.Note, n.Filename, n.Sticky
FROM notes n
INNER JOIN ItemNotes i ON i.NoteId = n.ID
WHERE i.AccountID = 5
ORDER BY TStamp DESC
```

Properties

Appearance

Font
Font of text of this component
Scripting name: `font`
Data type: Font

Foreground Color
The foreground color of the component.
Scripting name: `foreground`
Data type: Color

Sticky Header Color
The background color of the header for sticky notes
Scripting name: `stickyHeaderColor`
Data type: Color

Sticky Note Color
The background color for stick notes
Scripting name: `stickyNoteColor`
Data type: Color

Header Color
The background color of the header notes
Scripting name: `headersColor`
Data type: Color

Note Color
The background color for notes
Scripting name: `noteColor`
Data type: Color

Date Format
The format string to use for the date of the note.
Scripting name: `dateFormat`
Data type: String

Padding
The amount of padding between the notes.
Scripting name: `padding`
Data type: int

Behavior

Database Connection
Name of the database connection to run the queries against. Leave blank to use project's default connection.
Scripting name: `datasource`
Data type: String

Insert Query 1
This insert query will insert a new note into a notes table.
The placeholder %s will be replaced with the current username.
The query will be run as a prepared statement, so the query also needs to accept parameters by using the ? placeholder.
If attachments are enabled it will use four parameters: Note Body, Attachment Bytes, Attachment Name, Sticky (0/1)
When attachments are disabled, it will use two parameters: Note Body, Sticky (0/1)

© 2011 Inductive Automation

Scripting name `insertQuery1`
Data type String

Insert Query 2
This optional insert query inserts the mapping for a new note into a mapping table.
%d will be replaced with the ID of the new note.
To disable this behavior, simply set this property to a blank string.

Scripting name `insertQuery2`
Data type String

Delete Query
This query is used for deleting a note. %d is replaced with the note's ID

Scripting name `deleteQuery`
Data type String

Unstick Query
This query is used for changing a note's status to be not sticky. %d is replaced with the note's ID

Scripting name `unstickQuery`
Data type String

Download Attachment Query
This query is used for downloading binary attachments. %d is replaced with the note's ID

Scripting name `getAttachmentQuery`
Data type String

Delete Mode
Controls if anyone can delete any note, noone can delete a note, or only owners can delete their notes

Scripting name `deleteMode`
Data type int
Flags expert
Values
0 No Deletes
1 Ow ner Deletes
2 Any Deletes

Attachments Enabled
Controls whether or not files can be attached to notes.

Scripting name `attachmentsEnabled`
Data type boolean

Download Mode
What to do when an attachment is downloaded.

Scripting name `downloadMode`
Data type int
Values
0 Save
1 Open

Touchscreen Mode
Controls when this input component responds if touchscreen mode is enabled.

Scripting name `touchscreenMode`
Data type int
Flags expert
Values
0 None
1 Single-Click
2 Double-Click

Common

Name
The name of this component.

Scripting name `name`
Data type String
Flags bindable

© 2011 Inductive Automation

Enabled — If disabled, a component cannot be used.
Scripting name: `enabled`
Data type: boolean

Visible — If disabled, the component will be hidden.
Scripting name: `visible`
Data type: boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name: `border`
Data type: Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name: `toolTipText`
Data type: String

Data

Data — Fill this DataSet in with the notes for the desired entity. Columns are: Id, Username, Timestamp, NoteBody, Filename, IsSticky
Scripting name: `data`
Data type: Dataset
Flags: bindable

Data Quality — The data quality code for any tag bindings on this component.
Scripting name: `dataQuality`
Data type: int
Flags: bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- propertyChange

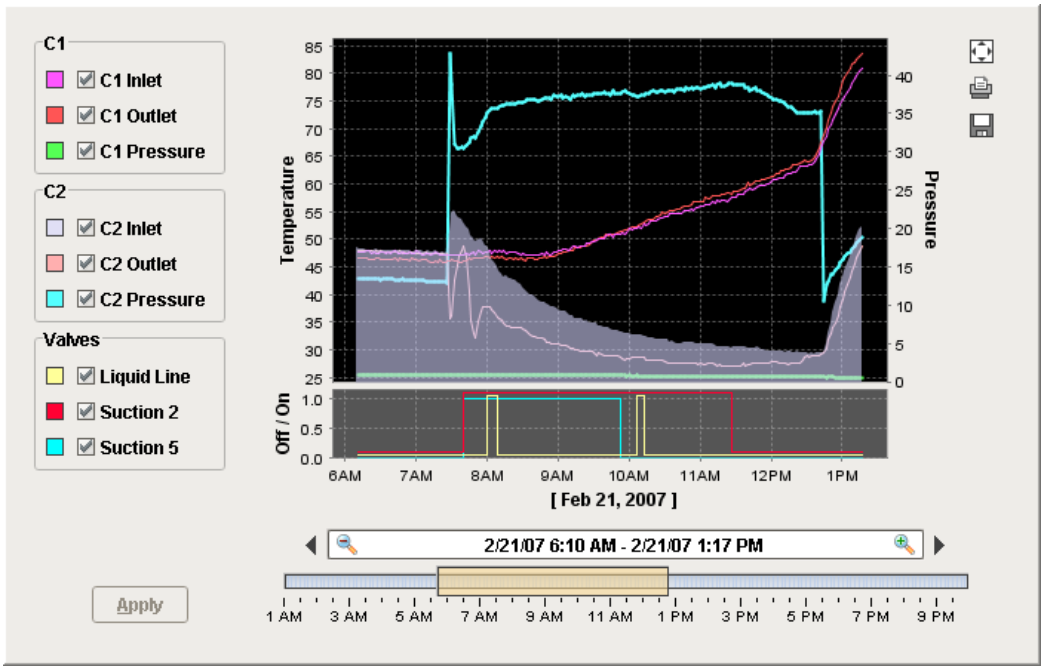
Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.5 Charts

7.5.1 Easy Chart



Description

This component is used to make powerful and runtime-configurable timeseries charts. It is configured by defining a set of *pens* and *axes*. Each pen represents a series of data. Pens can be many different styles, such as line, area, bar, and shape. This chart automatically creates controls for picking the time range and for hiding or displaying pens.

Features

Easy configuration
User-selectable set of pens
Automatic time-selection controls
SQL Query and/or SQLTags Historian data sources
Automatic SPC and calculated pen support
Zoom, Pan, X-Trace modes
Any number of Y-axes and subplots
Realtime or Historical

Pens

The are three kinds of pens in the Easy Chart:

1. **SQLTags Historian Pens**. These pens pull their data from the SQLTags Historian system.
2. **Database Pens.** These pens will automatically create SQL SELECT queries to pull data from a database table. Typically, this is a table that is the target of a Historical Transaction Group.
3. **Calculated Pens**. These pens display a calculated dataset based off another pen, such as a moving average or an SPC function such as the UCL (Upper Control Limit).

Modes: Realtime vs Historical

The Easy Chart can operate in 3 different modes. These modes affect the range of data that is displayed, the controls the user is shown, and whether or not the chart polls for data.

© 2011 Inductive Automation

1. **Historical Mode**. In this mode, the user is shown a Date Range component to pick the range of data to fetch and display. The initial values of this component are set through properties on the chart. In historical mode, the chart does not poll.
2. **Realtime Mode**. In this mode, the user is given the opportunity to pick the amount of time in the past to display. For example, the last 5 minutes or the last 2 hours. The chart will poll at a rate according to the *Poll Rate* parameter.
3. **Manual Mode**. In this mode, the chart will use the values if its *Start Date* and *End Date* parameters to govern what data is displayed. Polling is controlled by having the *Poll Rate* at zero (polling off) or greater than zero.

Basic Chart Configuration

The Easy Chart has many properties, like other components, that control its behavior. Things like its *Mode*, *Polling Rate*, etc are configured via the properties. All of the setup for adding pens, axes, subplots, etc is its Customizer. You can also drag and drop Historian-enabled SQLTags onto the chart directly in the Designer to add those tags as chart pens.

Y-Axes

The easy chart supports any number of Y-axes. To add an axis, go to the *Axes* tab of the chart customizer. When adding an axis, you get a number of options such as the type (numeric or logarithmic), label, color, autorange vs fixed range, and auto-ticks vs fixed ticks. You can also modify the position of the axis, but note that by default the Chart's *Auto Axis* Positioning property is enabled, which means that the chart will balance the axes automatically between left and right depending on demand. As pens are turned on and off by the user, only the axes that are used by visible pens are shown.

After you add your axes, you edit any pens that you want to use your new axes. Simply choose the new axis in the axis dropdown of the pen editing window.

Subplots

The Subplots feature lets you break up the chart's plot area into multiple distinct subplots that share the X axis, but have their own Y axes. This is often useful for digital data, as shown in the screenshot above. By default the chart has 1 subplot (the main plot). To add a new subplot, simply hit the add button in the *Subplots* tab of the chart customizer.

Subplots have relatively few options. The *Weight* option determines how much room the subplot gets relative to the other subplots. For example, in the screenshot above subplot #1's weight is 5, and subplot #2's weight is 1, leading to a 5-to-1 distribution of space. Just like axes, once you add your subplots you should go back to your pens and modify you pens' subplot property for any pens you want to appear on the subplot.

Pen Groups

You can put your pens in groups to break up the pens into some logical separation. For instance, in the screenshot above there are three pen groups: C1, C2, and Valves. The group name is used as the titled border for the pens' grouping container. Groups also have another purpose, but it is more advanced and most people won't have to worry about it. For more, read the Dynamic Pens section below.

Advanced Configuration

Dynamic Pens

In is often the case that you'll want to make one chart window that services many similar pieces of equipment. For instance, if you have 30 tanks and they all have the same datapoints, you want to be

© 2011 Inductive Automation

able to use one window for all 30 of them and simply pass the tank number into the chart window as a parameter. There are actually a number of ways to accomplish this, each method suitable for different scenarios.

Database pens have 2 ways to be made dynamic. The first is the Chart's *Where Clause* property. This is a snippet of SQL where clause syntax, like "`machine_num = 28`" that will be included for *all* database pens in their queries. The second is to use a *dynamic group.* Any group can be made a dynamic group in the customizer. For each dynamic group, the easy chart will get a special dynamic property associated with that group. That property is another snippet of SQL where clause that will be applied to all database pens in that group.

The other way to make your pens (and anything else about the chart) dynamic at runtime is to use *dynamic configuration*. Read on...

Dynamic Configuration
The Easy Chart is not just meant to be easy to configure, but also very powerful. In particular, there is an emphasis on the ability to make any configuration change dynamically in a client - not just statically in the Designer. While a bit of scripting or clever property binding may be required, the technique is very powerful. This is achieved by storing all of the settings that you alter in the customizer in a set of expert-level dataset properties. So altering the datasets alters the chart configuration. You can inspect these various datasets, which hold the pens, axes, and subplot information, to see their format. They all look up information by column name (case-insensitive). So, if you have pen configuration stored in a database, you can bind an indirect SQL Query binding to alter the chart's pen set at runtime.

Properties

Appearance

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the component.

Scripting name	`background`
Data type	Color

Plot Background — The backgroud color for all plots, unless they override it

Scripting name	`plotBackground`
Data type	Color

Gridline Color — The color of the gridlines.

Scripting name	`gridlineColor`
Data type	Color

Gridline Width — The width (thickness) of the gridlines.

Scripting name	`gridlineWidth`
Data type	float

Gridline Dash Pattern — The dash pattern for the gridlines.

Scripting name	`gridlineDashPattern`
Data type	String

Border — The border surrounding the entire chart component.

Scripting name	`border`
Data type	Border

© 2011 Inductive Automation

Chart Border
The border for the chart itself
Scripting name `chartBorder`
Data type Border

Pen Control Border
The border for the pen control panel, if visible
Scripting name `penBorder`
Data type Border

Date Range Border
The border for the date range control, if visible
Scripting name `dateRangeBorder`
Data type Border

Chart Title
Sets an optional title to be displayed above the chart
Scripting name `title`
Data type String

X Axis Label
The label shown on the X Axis (time axis)
Scripting name `xAxisLabel`
Data type String

Font
Font of text of this component
Scripting name `font`
Data type Font

Axis Font
The font for axis labels
Scripting name `axisLabelFont`
Data type Font

Tick Font
The font for tick labels
Scripting name `axisTickLabelFont`
Data type Font

Behavior

Chart Mode
Affects the mode that the chart operates in.Manual Mode: the data selected is determined by the values of the Start Date and End Date properties, which must be set manually.Historical Mode: a date range component will be displayed by the chart, allowing the user to select the time peried they are interested inRealtime Mode: the user will be given the change to choose a span of time, like 15 minutes, and that span will be updated at the poll rate as the data scrolls across
Scripting name `chartMode`
Data type int
Flags bindable
Values
0 Manual
1 Historical
2 Realtime

Pen Control?
Controls whether or not end-users can turn on and off pens.
Scripting name `allowPenManipulation`
Data type boolean

Pen Control Mode
The style in which the pen control panel alters the chart configuration. In heavyweight mode, unchecked pens are not queried, but checking and unchecking pens refreshes the chart. In lightweight mode, all pens are queried, but checking and unchecking pens is quick.
Scripting name `penControlMode`

© 2011 Inductive Automation

Data type	int	
Values	0	Heavyweight
	1	Lightweight

Auto Apply

If true, user changes to pen visibility will occur immediately.

Scripting name	autoApply
Data type	boolean

Poll Rate

The rate (in milliseconds) at which this chart's queries poll. Historical charts don't use this property.

Scripting name	pollRate
Data type	int

X Axis AutoRange?

If true, the X axis will automatically fit the range of available data, if false, it will display a fixed range based on the start date and end date.

Scripting name	xAxisAutoRange
Data type	boolean

X Axis Margin

A margin for the upper and lower ends of the x axis, expressed as a percentage of the total range.

Scripting name	xAxisMargin
Data type	double

Empty Group Name

The group name to use for pens that are not in a pen group.

Scripting name	emptyGroupName
Data type	String

Group Pens

If true, pens will be grouped by their group name

Scripting name	penGrouping
Data type	boolean

Auto Axis Positioning

If true, axes alternate automatically between left and right, rather than being placed explicitly.

Scripting name	autoPositionAxes
Data type	boolean
Flags	expert

Auto Pen Coloring

If true, pens are assigned different colors automatically.

Scripting name	autoColorPens
Data type	boolean
Flags	expert

Auto Color List

The list of colors to use if auto pen coloring is enabled

Scripting name	autoColorList
Data type	Color[]
Flags	expert

Show Loading

If true, an animated indicator will be shown when data is loading

Scripting name	showLoading
Data type	boolean

Show Warnings

If true, warnings generated during chart configuration will be printed to the console.

Scripting name	showWarnings
Data type	boolean
Flags	expert

Show Popup?

If true, a popup menu will be shown on right-click that allows the user to

© 2011 Inductive Automation

change mode, print, save, etc.

Scripting name `showPopup`
Data type boolean
Flags expert

Show Tooltips?

If true, tooltips showing point values will be displayed on the chart.

Scripting name `tooltips`
Data type boolean

Chart Configuration

DB Pens

This Dataset defines all of the database pens for the chart.

Scripting name `pens`
Data type Dataset
Flags bindable | expert

Tag Pens

This Dataset defines all of the SQLTag History pens for the chart.

Scripting name `tagPens`
Data type Dataset
Flags bindable | expert

Calculated Pens

This Dataset defines the calculated pens for the chart.

Scripting name `calcPens`
Data type Dataset
Flags bindable | expert

Axes

This Dataset defines all axis that can be used by the pens.

Scripting name `axes`
Data type Dataset
Flags expert

Subplots

This Dataset defines all subplots' relative size and color.

Scripting name `subplots`
Data type Dataset
Flags expert

Common

Name

The name of this component.

Scripting name `name`
Data type String
Flags bindable

Visible

If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor

The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Flags expert
Values
`0` Default
`1` Crosshair
`2` Text

© 2011 Inductive Automation

3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Selected X Value

The selected domain axis value for X-Trace and Mark modes.

Scripting name	`selectedXValue`
Data type	String
Flags	bindable | read-only

Tag History Resolution

The number of datapoints to request for tag history pens. -1 means automatic, which uses the width of the chart.

Scripting name	`tagHistoryResolution`
Data type	int
Flags	expert

Where Clause

A snippet of where clause that will be applied to all pens, like "TankNum = 2"

Scripting name	`globalWhereClause`
Data type	String

Start Date

For manual-mode. The start date to use for selecting pen data

Scripting name	`startDate`
Data type	Date
Flags	bindable

End Date

For manual-mode. The end date to use for selecting pen data

Scripting name	`endDate`
Data type	Date
Flags	bindable

Historical Range

Startup Range

For historical-mode date range. If startup mode is Automatic, this will be the starting range of time available for selection.

Scripting name	`startupRange`
Data type	String

Startup Selection

For historical-mode date range. If startup mode is Automatic, this will be the starting selected range.

Scripting name	`startupSelection`
Data type	String

Max Selection

For historical-mode date range. The maximum size of the selected date range

Scripting name	`maxSelectionSize`
Data type	String

Date Style

The style to display dates in. For international support.

Scripting name	`dateStyle`
Data type	int
Flags	expert

© 2011 Inductive Automation

Values
- 0 Auto
- 1 MDY
- 2 DMY
- 3 YMD

Time Style

The style to display times of day. For international support.

Scripting name: timeStyle
Data type: int
Flags: expert
Values:
- 15 Auto
- 16 12 HR
- 17 24 HR

Show Density

For historical-mode date range. If true, a data density histogram will be shown in the date range.

Scripting name: showHistogram
Data type: boolean

Today Color

For historical-mode date range. The color of the "Today Arrow" indicator

Scripting name: todayIndicatorColor
Data type: Color
Flags: expert

Box Fill

For historical-mode date range. The fill color for the selection box.

Scripting name: boxFill
Data type: Color
Flags: expert

Selection Highlight

For historical-mode date range. The focus highlight color for the selection box

Scripting name: selectionHighlight
Data type: Color
Flags: expert

Track Margin

For historical-mode date range. The amount of room on either side of the slider track. May need adjusting of default font is changed.

Scripting name: trackMargin
Data type: int
Flags: expert

Tick Density

For historical-mode date range. This is multiplied by the width to determine the current ideal tick unit.

Scripting name: tickDensity
Data type: float
Flags: expert

High Density Color

For historical-mode date range. The color used to indicate high data density.

Scripting name: highDensityColor
Data type: Color
Flags: expert

Layout

Date Range

Affects the position of the date range control.

Scripting name: dateRangeLocation
Data type: int
Values:
- 1 Top
- 2 Bottom

© 2011 Inductive Automation

Legend — Where the legend should appear, if any.

Scripting name	`legend`
Data type	int
Values	**0** None **1** Top **2** Bottom **3** Left **4** Right

Horiz Gap — The horizontal spacing to use for the pen checkboxes

Scripting name	`hGap`
Data type	int
Flags	expert

Vert Gap — The vertical spacing to use for the pen checkboxes

Scripting name	`vGap`
Data type	int
Flags	expert

Alphabetize Pens — If true, pens visibility checkboxs will be alphabetized.

Scripting name	`alphabetizePens`
Data type	boolean
Flags	expert

Pen Style Options

Bar Margin — The margin to use for the 'Bar' pen style

Scripting name	`barMargin`
Data type	double
Flags	expert

Gap Threshold — The relative threshold to use for determining continuity breaks for the 'Discontinous Line' pen style

Scripting name	`gapThreshold`
Data type	double
Flags	expert

3D X Offset — The offset to use in the x direction for the '3D Line' pen style

Scripting name	`xOffset3D`
Data type	int
Flags	expert

3D Y Offset — The offset to use in the y direction for the '3D Line' pen style

Scripting name	`yOffset3D`
Data type	int
Flags	expert

Digital Gap — The size of the gap to use between digital pens

Scripting name	`digitalGap`
Data type	double
Flags	expert

Realtime Range

Unit Count — For realtime-mode date range. The number of units back to display

Scripting name	`unitCount`
Data type	int

Unit — For realtime-mode date range. The selected unit of the realtime date

© 2011 Inductive Automation

control

Scripting name `unit`
Data type int
Values
`1` Seconds
`60` Minutes
`360` Hours
`0`
`864` Days
`00`

Realtime Text — For realtime-mode date range. The text to display on the realtime date control.

Scripting name `rtLabel`
Data type String

Uncategorized

Properties Loading — The number of properties currently being loaded

Scripting name `propertiesLoading`
Data type int
Flags bindable | read-only

Total Datapoints — The number of datapoints being displayed by the graph.

Scripting name `datapoints`
Data type int
Flags bindable | read-only

Utility Buttons

Show Maximize Button? — If true, a small maximize button will be displayed next to the chart.

Scripting name `showMaximize`
Data type boolean

Show Print Button? — If true, a small print button will be displayed next to the chart.

Scripting name `showPrint`
Data type boolean

Show Save Button? — If true, a small save button will be displayed next to the chart.

Scripting name `showSave`
Data type boolean

Button Size — The size of the utility button icons.

Scripting name `utilityButtonSize`
Data type int
Flags expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

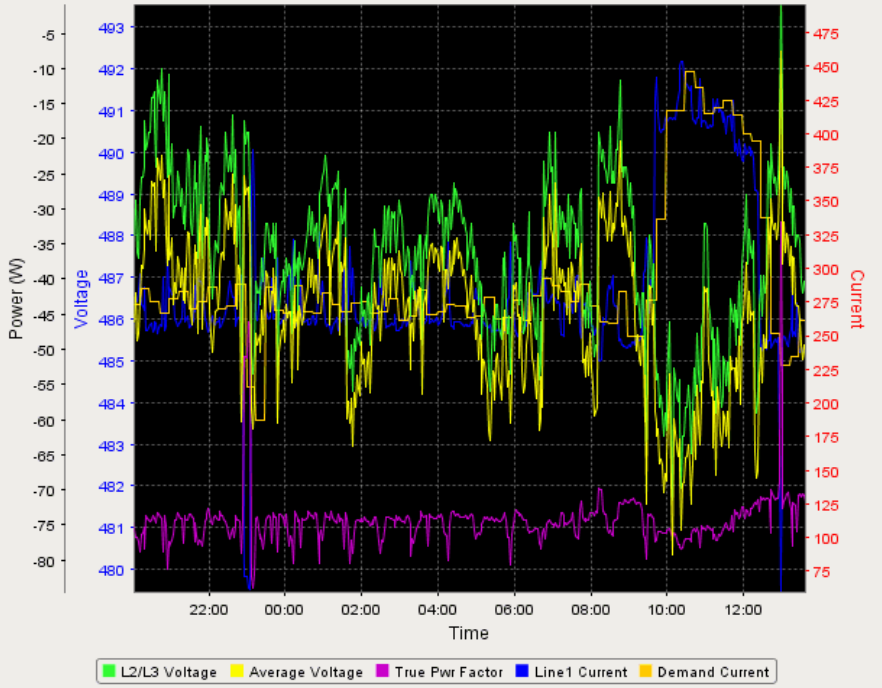
mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.5.2 Chart



Description

The Chart component (also called the *Classic Chart* when contrasted with the Easy Chart) provides a flexible way to display either timeseries or X-Y charts that are powered by any number of datasets. Typically, these datasets are bound to SQL Query bindings.

Features

SQL Query and/or SQLTags Historian data sources
Zoom, Pan, X-Trace modes
Any number of Y-axes and subplots
Realtime or Historical
Many different rendering styles

Configuration

The basic idea behind configuring the class chart is simple: add datasets, and fill them in with data in a format that the chart understands. You add datasets to the chart using the chart's customizer. You then use standard property binding to put data into these charts. Commonly you'll use a SQL Query binding. Since these datasets are just normal dynamic properties, you can also access them via scripting.

The Customizer also lets you add additional X and Y axes. There are various types of axes, and they each have a large number of properties. Lastly, you can configure additional properties for each dataset, such as which axes it maps to, its visual style, subplot, etc.

Datasets

Each dataset should define one or more "series" (a.k.a "pens"). The format for these datasets is quite simple. Each series in a dataset shares common X-values, defined by the first column. Each additional column are the Y-values for a series. For example:

© 2011 Inductive Automation

t_stamp	motor_amps	motor_speed	motor_hoa_state
2010-01-13 8:05:00	16.8	223	0
2010-01-13 8:10:00	16.8	245	0
2010-01-13 8:15:00	16.9	244	1

Note that it is certainly not a coincidence that this looks just like a database table that the Historical Group is logging to. It is also what the result datasets of a SQLTags Historian query looks like.

Rows *must be* sorted in ascending order. The chart will draw and connect the points in whatever order you provide, them, so unless you want a jumbled mess - don't forget the **ORDER BY** clause in your query.
Make sure that your timestamp column, as well as other columns that may appear in your WHERE clause, are *indexed*. This will help your chart queries run much faster. We've seen queries that literally take over 5 minutes of database-cranking reduced to a few seconds with the addition of an index.
String values are not allowed (except in category chart x-values, see below). Make sure your database columns are numeric, or date/time for x-values.

Binding Techniques
The classic chart can be used to make almost any kind of chart, with some effort. Historical, realtime, dynamic pen selection, etc is all possible. Your job is just to fill the datasets with the pertinent data, and the chart will display it.

The most common idea is to make the chart dynamic by varying the date range that the datas'ts SQL Query bindings run. This is easy to do by adding a Date Range component and using Indirect Bindings.

Chart Type: XY vs Category
The classic chart is typically in `XY Plot` mode. This means that the x-axis is either date or numeric, and the y-axes are numeric. If your x-axis is categorical (names, not numbers), you can switch the *Chart Type* property to `Category Chart`. Don't be surprised when you get a few errors - you'll need to go and switch your x-axis to be a Category Axis, and fill your dataset in with valid category data, that is, String-based x-values. This is most often used with the bar-renderer (see the Customizer).

Properties

Appearance

Font
Font of text of this component
Scripting name `font`
Data type Font

Foreground Color
The foreground color of the component.
Scripting name `foreground`
Data type Color

Background Color
The background color of the component.
Scripting name `background`
Data type Color

Plot Background
The backgroud color for all plots, unless they override it
Scripting name `plotBackground`
Data type Color

© 2011 Inductive Automation

Chart Title
An optional title that will appear at the top of the chart.
Scripting name `title`
Data type String

Chart Orientation
The orientation of the domain axis of the chart.
Scripting name `orientation`
Data type int
Values
0 Horizontal
1 Vertical

Show Legend?
If true, a legend will be shown for the series displayed in the chart.
Scripting name `legend`
Data type boolean

Selection Highlight Color
The color of the selection highlight
Scripting name `selectionHighlightColor`
Data type Color
Flags expert

Selection Highlight Width
The line width of the selection highlight
Scripting name `selectionHighlightWidth`
Data type float
Flags expert

Behavior

Chart Type
Choose the type for this chart: XY (Numeric X-axis) or Category (String X-axis)
Scripting name `chartType`
Data type int
Values
2 XY Plot
0 Category Chart

Extract Order
Extract order for how category datasets should be interpreted.
Scripting name `extractOrder`
Data type int
Values
0 By Col
1 By Row

Subplot Mode
The axis that subplots share if more than 1 subplot.
Scripting name `subplotMode`
Data type int
Values
0 Shared Domain
1 Shared Range

Show Tooltips?
If true, tooltips showing point values will be displayed.
Scripting name `tooltips`
Data type boolean

Show Popup?
If true, a popup menu will be shown on right-click that allows the user to change mode, print, save, etc.
Scripting name `showPopup`
Data type boolean
Flags expert

Selection Enabled?
If true, the user will be able to select datapoints on the chart. The selected datapoint will be highlighted, and the "selectedData" property will reflect it.

© 2011 Inductive Automation

Scripting name `selectionEnabled`
Data type boolean

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Flags expert
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Selected Datapoint — The currently selected datapoint

Scripting name `selectedData`
Data type String
Flags bindable | read-only

Selected X Value — The selected domain axis value for X-Trace and Mark modes.

Scripting name `selectedXValue`
Data type String
Flags bindable | read-only

Data Quality — The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Uncategorized

© 2011 Inductive Automation

Properties Loading — The number of properties currently being loaded

Scripting name	`propertiesLoading`
Data type	int
Flags	bindable | read-only

Scripting

Events

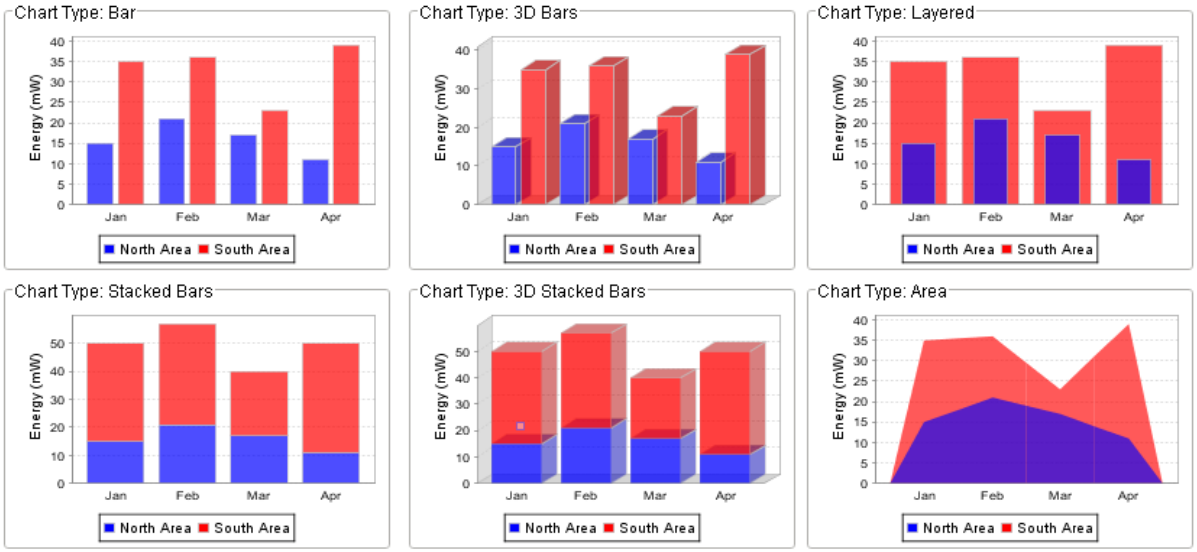
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.5.3 Bar Chart



Description

The Bar Chart is a very easy-to-use chart that provides familiar bar charts. It also can be configured to display other kinds of category charts. A category chart is a chart whose X-values are categories (strings) rather than numeric values (numbers, dates).

Like most chart components (other than the Easy Chart), the *Data* property drives the chart. The first column in the Data dataset defines the names of the categories. The rest of the columns define the values for each of the series (if there is more than one series per category), and thus should be numeric. Note - if your data is 'turned on its side', meaning that the columns define the categories and rows define the series, then set the Extract Order to "By Column".

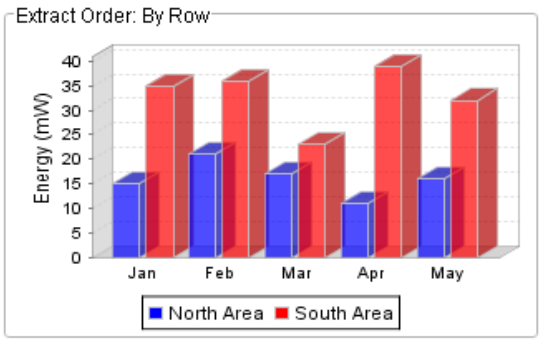
Extract Order Example

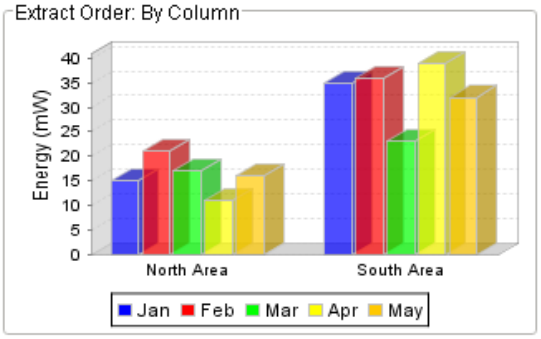
The following two charts demonstrate the effects of the extract order property on the given dataset

Label (String)	**North Area** (Integer)	**South Area** (integer)
Jan	15	35

© 2011 Inductive Automation

Feb	21	36
Mar	17	23
Apr	11	39
May	16	32





Properties

Appearance

Chart Title
An optional title that will appear at the top of the chart.

Scripting name `title`
Data type String

Chart Type
Controls how the bar chart is displayed.

Scripting name `rendererType`
Data type int
Values
- **0** Bar
- **1** 3D Bars
- **2** Stacked Bars
- **3** 3D Stacked Bars
- **4** Layered
- **5** Area

Plot Background
The backgroud color for the plot.

Scripting name `plotBackground`
Data type Color

Series Colors
The sequence of colors used for series in the bar chart.

Scripting name `seriesColors`
Data type Color[]

Legend?

Scripting name `legend`
Data type boolean

Labels?
Always display labels?

Scripting name `labels`
Data type boolean

Gradient bars?
If true, bars will be painted with a gradient 'shine'.

Scripting name `gradient`
Data type boolean

Shadows?
If true, bars will have a drop-shadow beneath them.

© 2011 Inductive Automation

Scripting name `shadows`
Data type boolean

Foreground Transparency The transparency of the pie (useful for 3D pies)

Scripting name `foregroundAlpha`
Data type float

Vertical Sets the orientation of the chart to vertical (true) or horizontal(false)

Scripting name `vertical`
Data type boolean

Category Margin The marign between categories as a fraction of the total space

Scripting name `categoryMargin`
Data type double

Item Margin The margin between bars in a category as a fraction

Scripting name `itemMargin`
Data type double

Axes

Value Axis Label The label for the value axis

Scripting name `valueLabel`
Data type String

Category Axis Label The label for the category axis

Scripting name `categoryLabel`
Data type String

Value Axis Auto-Range If true, the value axis range will be determined automatically. If false, the specified upper and lower bounds will be used.

Scripting name `valAxisAutoRange`
Data type boolean

Value Axis Lower Bound The lower bound of the value axis. Used only when auto-range is false.

Scripting name `valAxisLowerBound`
Data type double

Value Axis Upper Bound The upper bound of the value axis. Used only when auto-range is false.

Scripting name `valAxisUpperBound`
Data type double

Category Axis Label Angle The angle for the value axis' labels.

Scripting name `catAxisLabelPosition`
Data type int
Values
- **0** Standard
- **1** Dow n 45
- **2** Dow n 90
- **3** Up 45
- **4** Up 90

Title Font The font for the chart's title.

Scripting name `titleFont`
Data type Font

Legend Font The font for the legend items.

Scripting name `legendFont`
Data type Font

© 2011 Inductive Automation

Bar Label Font	The font for the bar labels. Scripting name `barLabelFont` Data type Font
Bar Label Offset	The offset between the bar and the bar label. Scripting name `barLabelOffset` Data type double Flags expert
Value Axis Label Font	The font for the value axis label. Scripting name `valAxisLabelFont` Data type Font
Category Axis Label Font	The font for the category axis label. Scripting name `catAxisLabelFont` Data type Font
Value Axis Tick Font	The font for the value axis' ticks. Scripting name `valAxisTickFont` Data type Font
Category Axis Tick Font	The font for the category axis' ticks. Scripting name `catAxisTickFont` Data type Font
Bar Label Color	The color for the bar labels. Scripting name `barLabelColor` Data type Color
Value Axis Label Color	The color for the value axis label. Scripting name `valAxisLabelColor` Data type Color
Category Axis Label Color	The color for the category axis label. Scripting name `catAxisLabelColor` Data type Color
Value Axis Tick Color	The color for the value axis' ticks. Scripting name `valAxisTickColor` Data type Color
Category Axis Tick Color	The color for the category axis' ticks. Scripting name `catAxisTickColor` Data type Color
Value Axis Upper Margin	The upper margin, as a percentage, of the value axis. Only used when auto-range is true. Scripting name `valAxisUpperMargin` Data type double
Category Axis Upper Margin	The upper margin, as a percentage, of the category axis. Scripting name `catAxisUpperMargin` Data type double
Category Axis Lower Margin	The lower margin, as a percentage, of the category axis. Scripting name `catAxisLowerMargin` Data type double

© 2011 Inductive Automation

Behavior

Tooltips?

Scripting name `tooltips`
Data type boolean

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Enabled — If disabled, a component cannot be used.

Scripting name `enabled`
Data type boolean

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Flags expert
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Data — The data driving the chart.

Scripting name `data`
Data type Dataset

Data Quality — The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

© 2011 Inductive Automation

Extract Order	Controls whether the first row defines the categories or the series
Scripting name	`extractOrder`
Data type	int
Values	`0` By Column `1` By Row

Scripting

Events

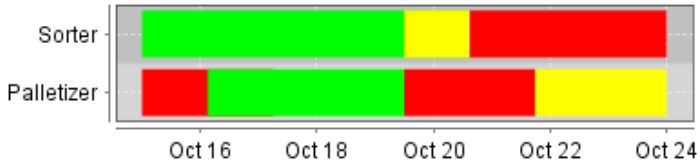
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.5.4 Status Chart



Description

The status chart component allows you to visualize the status of one or more discrete datapoints over a time range. The X-axis is always a timeseries axis, and the Y-axis is a category axis, with one entry per data series. The chart is populated with a single dataset, the first column of which must be a datetime column.

Wide vs Tall Datasets.

In `Wide` format, all of the columns but the first must be numeric. These "series" columns' headers will be used as the names on the y-axis. In Tall format, there should be exactly 3 columns. The first is the timestamp, the second is the series name, and the third is the value. For example:

Wide Format			Tall Format		
t_stamp	Valve1	Valve	t_stamp	Name	Value
2010-01-13 8:00:00	0	2	2010-01-13 8:00:00	Valve1	0
2010-01-13 8:02:00	0	2	2010-01-13 8:00:00	Valve2	2
2010-01-13 8:04:00	1	2	2010-01-13 8:02:00	Valve1	0
2010-01-13 8:06:00	1	1	2010-01-13 8:02:00	Valve2	2
2010-01-13 8:08:00	0	1	2010-01-13 8:04:00	Valve1	1
			2010-01-13 8:04:00	Valve2	2
			2010-01-13 8:06:00	Valve1	1
			2010-01-13 8:06:00	Valve2	1
			2010-01-13 8:08:00	Valve1	0
			2010-01-13 8:08:00	Valve2	1

Color Mapping

Apart from getting the data into the series chart, the only other commonly configured option is the mapping of discrete values to colors. This is done in the Series Chart Customizer. Each named

© 2011 Inductive Automation

series can have its own mapping of colors, if desired. These mappings are stored in the expert-level dataset property *Series Properties Data* so they can be altered at runtime.

Properties

Appearance

Background Color

The background color of the component.

Scripting name	`background`
Data type	Color

Chart Title

Title of this chart.

Scripting name	`chartTitle`
Data type	String

Title Font

Font of the chart title.

Scripting name	`titleFont`
Data type	Font

Title Color

Color of the chart title.

Scripting name	`titleColor`
Data type	Color

Series Spacing

Affects the amount of spacing between series. Can be between 0.0 and 1.0. The series present on this chart are given equal space to display themselves. Series spacing is the precentage of that space that they use to do so.

Scripting name	`seriesSpacing`
Data type	double

Date Style

The style to display dates in. For international support.

Scripting name	`dateStyle`
Data type	int
Flags	expert
Values	**0** Auto **1** MDY **2** DMY **3** YMD

Time Style

The style to display times of day. For international support.

Scripting name	`timeStyle`
Data type	int
Flags	expert
Values	**15** Auto **16** 12 HR **17** 24 HR

Common

Name

The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled

If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible

If disabled, the component will be hidden.

© 2011 Inductive Automation

Scripting name `visible`
Data type boolean

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor

The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Flags expert
Values
- 0 Default
- 1 Crosshair
- 2 Text
- 3 Wait
- 12 Hand
- 13 Move
- 4 SW Resize
- 5 SE Resize
- 6 NW Resize
- 7 NE Resize
- 8 N Resize
- 9 S Resize
- 10 W Resize
- 11 E Resize

Data

Data Format

Format of the incoming data. In "wide" format, the first column of the dataset needs to be a timestamp, and every subsequent column represents one series in the chart. In "tall" format, the first column is a timestamp, the second column is a series name, and the third a value.

Scripting name `dataFormat`
Data type int
Values
- 0 Wide
- 1 Tall

Series Data

Data about each series. Data can be in either "wide" or "tall" format.

Scripting name `data`
Data type Dataset
Flags bindable

Series Properties Data

Properties for each series

Scripting name `properties`
Data type Dataset
Flags bindable | expert

Data Quality

The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Domain Axis

© 2011 Inductive Automation

Domain Axis Label — Label on the domain axis.

Scripting name: `domainAxisLabel`
Data type: String

Domain Axis Font — Font used on the domain axis.

Scripting name: `domainAxisFont`
Data type: Font

Domain Axis Color — Color used on the domain axis.

Scripting name: `domainAxisColor`
Data type: Color

Domain Axis Location — Location of the domain axis.

Scripting name: `domainAxisLocation`
Data type: int
Values: 2 Left; 3 Right

Show Domain Axis — Sets whether or not the domain axis is visible

Scripting name: `domainAxisVisible`
Data type: boolean

Range Axis

Range Axis Label — Label on the range axis.

Scripting name: `rangeAxisLabel`
Data type: String

Range Axis Font — Font used on the range axis.

Scripting name: `rangeAxisFont`
Data type: Font

Range Axis Color — Color used on the range axis.

Scripting name: `rangeAxisColor`
Data type: Color

Range Axis Location — Location of the range axis.

Scripting name: `rangeAxisLocation`
Data type: int
Values: 0 Top; 1 Bottom

Range Axis Lower Margin — Lower margin of the range axis.

Scripting name: `rangeAxisLowerMargin`
Data type: double

Range Axis Upper Margin — Upper margin of the range axis.

Scripting name: `rangeAxisUpperMargin`
Data type: double

Show Range Axis — Sets whether or not the range axis is visible.

Scripting name: `rangeAxisVisible`
Data type: boolean

Uncategorized

Properties Loading — The number of properties currently being loaded

Scripting name: `propertiesLoading`
Data type: int

© 2011 Inductive Automation

Flags	bindable | read-only

Scripting

Events

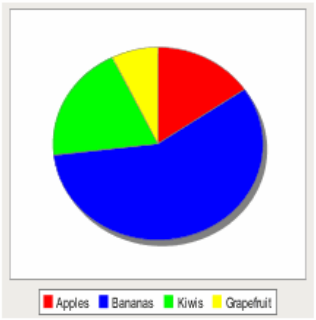
The following event sets are fired by this component. See Component Event Handlers to learn more.

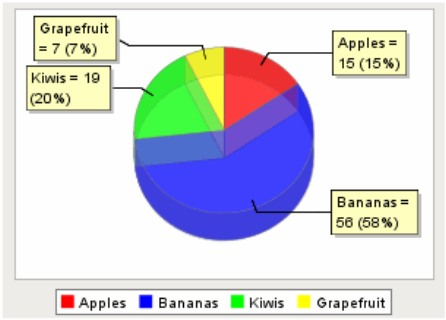
mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.5.5 Pie Chart





Description

The Pie Chart component displays a familiar-looking pie chart. A Pie Chart displays a list of named items, each of which has a value that is part of a total. The total is the sum of the value of each item. The key to the Pie Chart component is the *Data* property, which contains the items that will be displayed as pie wedges. Typically, this dataset will be bound to a SQL Query Binding to pull dynamic data out of an external database.

Extract Order

Similar to other charts, the pie chart can actually accept data in two formats. You can tell the pie chart which format to use via its *Extract Order* property. The two extract orders are `By Column` or `By Row`. The following table shows the two styles for the data that created the pie chart in the screenshot.

By Column		By Row			
Label	**Value**	**Grapefruit**	**Apples**	**Bananas**	**Kiwis**
Grapefruit	7	7	15	56	19
Apples	15				
Bananas	56				
Kiwis	19				

Labels

In addition to the color-coded legend, the pie chart can annotate each wedge with a label. The format of the label is controlled via the *Label Format* property. For example, the format string used in the screenshot is "`{0} = {2} ({3})`"
This is a pattern string that uses the following placeholders:

© 2011 Inductive Automation

`{0}` - the item label
`{1}` - the item value
`{2}` - the item percentage

Properties

Appearance

Chart Title
An optional title that will appear at the top of the chart.
Scripting name `title`
Data type String

Plot Background
The backgroud color for all plots, unless they override it
Scripting name `plotBackground`
Data type Color

Section Colors
The colors to use for the pie wedge fills.
Scripting name `sectionColors`
Data type Color[]

Outline Colors
The colors to use for the pie wedge outlines.
Scripting name `outlineColors`
Data type Color[]

Outline Stroke
The width for the section outline stroke.
Scripting name `outlineStroke`
Data type float

Legend?
Should there be an item legend below the chart?
Scripting name `legend`
Data type boolean

Labels?
Should labels be displayed near sections?
Scripting name `labels`
Data type boolean

Label Format
Formatting String. '{0}' is the wedge name, '{1}' is the value, '{2}' is the percent.
Scripting name `labelFormat`
Data type String

Tooltip Format
Formatting String. '{0}' is the wedge name, '{1}' is the value, '{2}' is the percent.
Scripting name `tooltipFormat`
Data type String

Legend Font
The font for legend items, if there is a legend.
Scripting name `legendFont`
Data type Font

Label Font
The font for labels items, if there are labels.
Scripting name `labelFont`
Data type Font

Starting Angle
The start angle to draw the pie wedges.
Scripting name `startAngle`
Data type int

© 2011 Inductive Automation

Rotation
Draw the wedges clockwise or counter-clockwise from the starting angle?

Scripting name	`rotation`
Data type	int
Values	0 Clockwise 1 Counter-Clockwise

Enforce Circularity?
If true, the pie cannot be an oval, even if the overall chart is.

Scripting name	`circular`
Data type	boolean

Style
Style of pie chart, standard, 3D, or ring.

Scripting name	`style`
Data type	int
Values	0 Pie 1 3D Pie 2 Ring

3D?
Deprecated. Use Style property instead.

Scripting name	`threeDimensional`
Data type	boolean
Flags	expert

Foreground Transparency
The transparency of the pie (useful for 3D pies)

Scripting name	`foregroundAlpha`
Data type	double

3D Depth Factor
The depth of a 3D pie as a factor of the chart height

Scripting name	`depthFactor`
Data type	double

Selection Highlight Color
The color of the selection highlight

Scripting name	`selectionHighlightColor`
Data type	Color
Flags	expert

Selection Highlight Width
The line width of the selection highlight

Scripting name	`selectionHighlightWidth`
Data type	float
Flags	expert

Behavior

Tooltips?
Should tooltips be displayed when the mouse hovers over sections?

Scripting name	`tooltips`
Data type	boolean

Selection Enabled?
If true, the user will be able to select wedges on the chart. The selected wedge will be highlighted, and the "selectedData" property will reflect it.

Scripting name	`selectionEnabled`
Data type	boolean

Common

Name
The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

© 2011 Inductive Automation

Enabled
If disabled, a component cannot be used.
Scripting name enabled
Data type boolean

Visible
If disabled, the component will be hidden.
Scripting name visible
Data type boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name border
Data type Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name toolTipText
Data type String

Cursor
The mouse cursor to use when hovering over this component.
Scripting name cursorCode
Data type int
Flags expert
Values
0 Default
1 Crosshair
2 Text
3 Wait
12 Hand
13 Move
4 SW Resize
5 SE Resize
6 NW Resize
7 NE Resize
8 N Resize
9 S Resize
10 W Resize
11 E Resize

Data

Data
The data driving the chart.
Scripting name data
Data type Dataset
Flags bindable

Extract Order
Controls whether or not a pie plot views columns as pies, or rows.
Scripting name extractOrder
Data type int
Values
0 By Column
1 By Row

Selected Wedge
The currently selected wedge
Scripting name selectedData
Data type String
Flags bindable | read-only

Data Quality
The data quality code for any tag bindings on this component.
Scripting name dataQuality
Data type int
Flags bindable | expert

© 2011 Inductive Automation

Scripting

Events

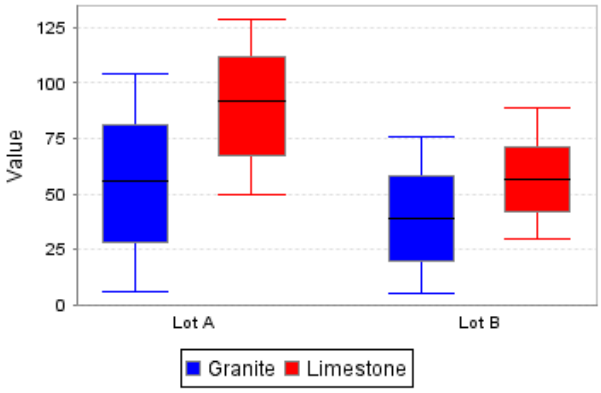
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.5.6 Box and Whisker Chart



Description

A Box and Whisker chart displays pertinent statistical information about sets of data. Each box represents a set of numbers. The upper and lower bounds of the box represent the 1st and 3rd quartiles. The line inside the box represents the median. The extends of the "whiskers" represent the max and min outliers. For a more detailed description, see http://mathworld.wolfram.com/Box-and-WhiskerPlot.html.

The configuration for setting up a box and whisker chart, like most charts, is populating the *Data* property. The dataset for a box and whisker chart contains sets of numbers. Each column defines a *series* of values, for which a "box" will be calculated. The column headers define the name for the box. You may also have an optional first column that is a String column, which can break up the series into categories. For example, the data that generated the plot in the screenshot would have looked like this:

Key (String)	**Granite** (Integer)	**Limestone** (Integer)
Lot A	23	39
Lot A	24	23
Lot A	93	54
Lot A	76	72
Lot B	21	83
Lot B	4	21
Lot B	76	98
Lot B	89	102

Properties

© 2011 Inductive Automation

Appearance

Font
Font of text of this component
Scripting name `font`
Data type Font

Chart Title
An optional title that will appear at the top of the chart.
Scripting name `title`
Data type String

Value Axis Title
A text label to display on the value axis.
Scripting name `valueAxisTitle`
Data type String

Category Axis Title
A text label to display on the category axis.
Scripting name `categoryAxisTitle`
Data type String

Series Colors
The colors to paint each box in a series.
Scripting name `seriesColors`
Data type Color[]

Plot Background
The backgroud color for the plot.
Scripting name `plotBackground`
Data type Color

Fill Boxes?
Fill the boxs with their color?
Scripting name `fillBoxes`
Data type boolean

Legend?
Show a legend on the chart?
Scripting name `legend`
Data type boolean

Behavior

Tooltips?
Show tooltips on tasks?
Scripting name `tooltips`
Data type boolean

Common

Name
The name of this component.
Scripting name `name`
Data type String
Flags bindable

Visible
If disabled, the component will be hidden.
Scripting name `visible`
Data type boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name `border`
Data type Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name `toolTipText`

© 2011 Inductive Automation

Data type String

Cursor
The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Flags expert
Values
0 Default
1 Crosshair
2 Text
3 Wait
12 Hand
13 Move
4 SW Resize
5 SE Resize
6 NW Resize
7 NE Resize
8 N Resize
9 S Resize
10 W Resize
11 E Resize

Data

Data
The data driving the chart.

Scripting name `data`
Data type Dataset

Data Quality
The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Scripting

Events

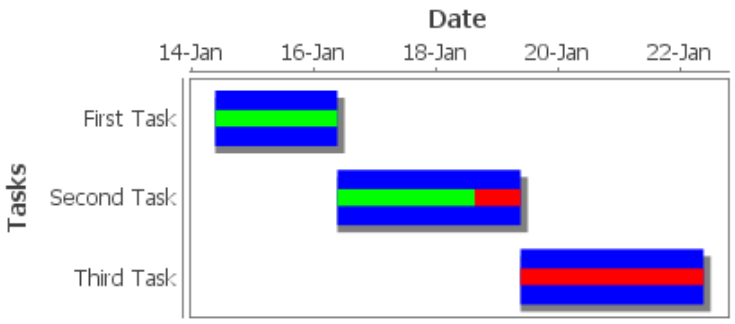
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.5.7 Gantt Chart



Description

A Gantt chart is used for task scheduling. It shows a list of named tasks, each of which have a start date, an end date, and a percentage complete. This allows an easy way to visualize tasks, workflows, and scheduling.

The Gantt chart is configured by populating its *Data* property. Each row of the dataset represents a task. There should be four columns: the task label, the start date, the end date, and the percentage (0-100) complete.

Properties

Appearance

Chart Title — An optional title that will appear at the top of the chart.

Scripting name	`title`
Data type	String

Task Axis Title

Scripting name	`taskAxisTitle`
Data type	String

Date Axis Title

Scripting name	`dateAxisTitle`
Data type	String

Task Color — The main color to draw tasks

Scripting name	`taskColor`
Data type	Color

Complete Color — The color to draw the amount completed in.

Scripting name	`completeColor`
Data type	Color

Incomplete Color — The color to draw the amount remaining to do in.

Scripting name	`incompleteColor`
Data type	Color

Plot Background — The backgroud color for the plot.

Scripting name	`plotBackground`
Data type	Color

Behavior

Tooltips? — Show tooltips on tasks?

Scripting name	`tooltips`
Data type	boolean

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled — If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible — If disabled, the component will be hidden.

Scripting name	`visible`

© 2011 Inductive Automation

Data type boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name `border`
Data type Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name `toolTipText`
Data type String

Cursor
The mouse cursor to use when hovering over this component.
Scripting name `cursorCode`
Data type int
Flags expert
Values
`0` Default
`1` Crosshair
`2` Text
`3` Wait
`12` Hand
`13` Move
`4` SW Resize
`5` SE Resize
`6` NW Resize
`7` NE Resize
`8` N Resize
`9` S Resize
`10` W Resize
`11` E Resize

Data

Data
The data driving the chart.
Scripting name `data`
Data type Dataset

Data Quality
The data quality code for any tag bindings on this component.
Scripting name `dataQuality`
Data type int
Flags bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

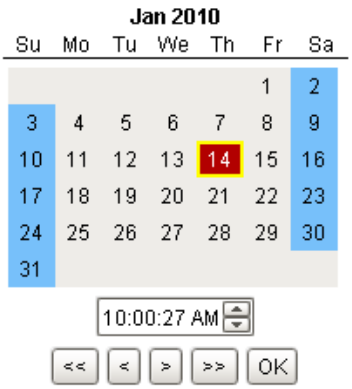
Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.6 Calendar

7.6.1 Calendar



Description

Displays a calendar and time input directly embedded in your window. Most commonly used by including one of the two date properties (immediate or latched) from the calendar in dynamic SQL Query Bindings.

Properties

Appearance

Font
Font of text of this component
Scripting name `font`
Data type Font

Foreground Color
The foreground color of the component.
Scripting name `foreground`
Data type Color

Background Color
The background color of the component.
Scripting name `background`
Data type Color

Today Foreground
Foreground color for the today indicator.
Scripting name `todayForeground`
Data type Color

Today Background
Background color for the today indicator.
Scripting name `todayBackground`
Data type Color

Weekend Foreground
Foreground color for the weekend indicators.
Scripting name `weekendForeground`
Data type Color

Weekend Background
Background color for the weekend indicators.
Scripting name `weekendBackground`
Data type Color

Selected Border
The border for the selected day indicator.
Scripting name `selectedBorder`

© 2011 Inductive Automation

Data type Border

Title Background
The background of the title bar
Scripting name titleBackground
Data type Color

Styles
Contains the component's styles
Scripting name styles
Data type Dataset
Flags bindable | expert

Behavior

Time Style
Select how this calendar should treat the time portion of the date.
Scripting name timeStyle
Data type int
Values
0 User Selectable
1 Start of Day
2 End of Day

Show OK Button
Turn this off if you don't want to show the OK button. The latched date and the immediate date will be equivalent.
Scripting name showOkButton
Data type boolean

Show Time
Turn this off if you don't want to show the time panel.
Scripting name showTime
Data type boolean

Format String
The date formatting pattern used to format the string versions of the dates.
Scripting name format
Data type String

Common

Name
The name of this component.
Scripting name name
Data type String
Flags bindable

Enabled
If disabled, a component cannot be used.
Scripting name enabled
Data type boolean

Visible
If disabled, the component will be hidden.
Scripting name visible
Data type boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name border
Data type Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name toolTipText
Data type String

© 2011 Inductive Automation

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name	`opaque`
Data type	boolean

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default **1** Crosshair **2** Text **3** Wait **12** Hand **13** Move **4** SW Resize **5** SE Resize **6** NW Resize **7** NE Resize **8** N Resize **9** S Resize **10** W Resize **11** E Resize

Data

Date (immediate) — The date as it is selected right now.

Scripting name	`date`
Data type	Date
Flags	bindable

Date (latched) — The date the last time "OK" was pressed.

Scripting name	`latchedDate`
Data type	Date
Flags	bindable

Formatted Date — The date property, as formatted by the format string.

Scripting name	`formattedDate`
Data type	String
Flags	bindable

Formatted Latched Date — The latched date property, as formatted by the format string.

Scripting name	`formattedLatchedDate`
Data type	String
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

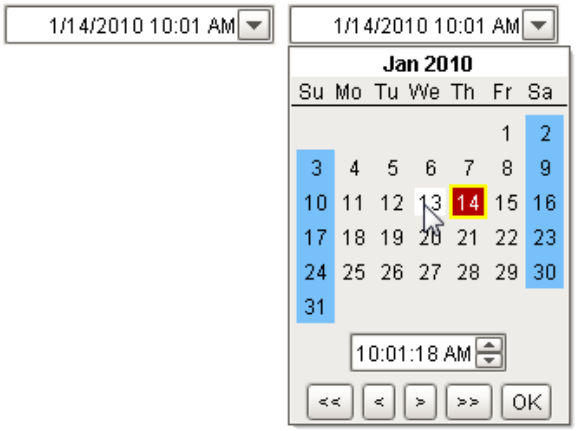
- mouse
- mouseMotion
- propertyChange

Scripting Functions

© 2011 Inductive Automation

This component has no special scripting functions.

7.6.2 Popup Calendar



A Popup Calendar in both collapsed and popup states

Description

The popup calendar is a popular way to provide date/time choosing controls on a window. Similar to the Calendar component, but takes up much less screen real estate. Most commonly used by including this component's *Date* property in dynamic SQL Query Bindings.

Properties

Appearance

Font	Font of text of this component Scripting name `font` Data type Font
Foreground Color	The foreground color of the component. Scripting name `foreground` Data type Color
Background Color	The background color of the component. Scripting name `background` Data type Color
Today Foreground	Foreground color for the today indicator. Scripting name `todayForeground` Data type Color
Today Background	Background color for the today indicator. Scripting name `todayBackground` Data type Color
Weekend Foreground	Foreground color for the weekend indicators. Scripting name `weekendForeground` Data type Color
Weekend Background	Background color for the weekend indicators. Scripting name `weekendBackground` Data type Color

© 2011 Inductive Automation

Selected Border
The border for the selected day indicator.
Scripting name `selectedBorder`
Data type Border

Title Background
The background of the title bar
Scripting name `titleBackground`
Data type Color

Calendar Background
The background color for the popup calendar
Scripting name `calendarBackground`
Data type Color

Styles
Contains the component's styles
Scripting name `styles`
Data type Dataset
Flags bindable | expert

Behavior

Time Style
Select how this calendar should treat the time portion of the date.
Scripting name `timeStyle`
Data type int
Values
0 User Selectable
1 Start of Day
2 End of Day

Show OK Button
Turn this off if you don't want to show the OK button. The latched date and the immediate date will be equivalent.
Scripting name `showOkButton`
Data type boolean

Show Time
Turn this off if you don't want to show the time panel.
Scripting name `showTime`
Data type boolean

Format String
The date formatting pattern used to display this date.
Scripting name `format`
Data type String

Common

Name
The name of this component.
Scripting name `name`
Data type String
Flags bindable

Enabled
If disabled, a component cannot be used.
Scripting name `enabled`
Data type boolean

Visible
If disabled, the component will be hidden.
Scripting name `visible`
Data type boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name `border`
Data type Border

© 2011 Inductive Automation

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default **1** Crosshair **2** Text **3** Wait **12** Hand **13** Move **4** SW Resize **5** SE Resize **6** NW Resize **7** NE Resize **8** N Resize **9** S Resize **10** W Resize **11** E Resize

Data

Date — The date that this component represents

Scripting name	`date`
Data type	Date
Flags	bindable

Text — The displayed text of the date (depends on the format string).

Scripting name	`text`
Data type	String
Flags	bindable | expert

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

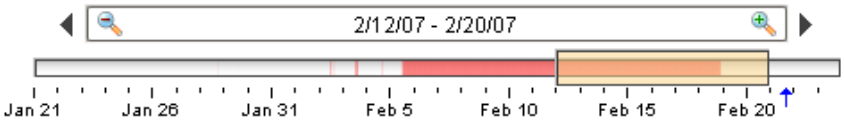
mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.6.3 Date Range



Description

The date range component provides an intuitive, drag-and-drop way to select a contiguous range of time. The user is shown a timeline and can drag or stretch the selection box around on the timeline. The selected range is always a whole number of units, where the unit is determined by the current zoom level. For instance, in the screenshot the selected range is Feb 12, 2007 through Feb 20, 2007. This means from the *beginning* of the 12th through the *end* of the 20th.

Using this component is as simple as using the *Start Date* and *End Date* properties that the component provides. Typically, you'll include these dates in a dynamic SQL query binding that drives whatever you're using the date range for, such as a table or chart. For instance, your query binding might look like this:

```
SELECT Column1, Column2, Column3
FROM MyTable WHERE
  t_stamp >= {Root Container.Date Range.startDate} AND
  t_stamp <= {Root Container.Date Range.endDate}
```

Data Density Histogram

As an advanced optional feature, the date range can display a *data density histogram* inside the timeline. This is useful for historical data with gaps in it, so that the end user isn't hunting for data. (Tip: this is also great for demos, to make it easy to find historical data in a database that isn't being continously updated).

To use this feature, bind the *Data Density* dataset to a query that returns just the timestamps of the target table. These timestamps will be used to fill in the histogram behind the timeline. You can use the *Outer Range Start Date* and *Outer Range End Date* properties in your query to limit the overall return size for the query.

Properties

Appearance

Font	Font of text of this component	
	Scripting name	`font`
	Data type	Font
Foreground Color	The foreground color of the component.	
	Scripting name	`foreground`
	Data type	Color
Background Color	The background color of the component.	
	Scripting name	`background`
	Data type	Color
Today Color	The color of the "Today Arrow" indicator	
	Scripting name	`todayIndicatorColor`
	Data type	Color

© 2011 Inductive Automation

Editor Background

The background color of the textual date range editor portion of this component.

Scripting name	`editorBackground`
Data type	Color

Box Fill

The fill color for the selection box.

Scripting name	`boxFill`
Data type	Color
Flags	expert

Selection Highlight

The focus highlight color for the selection box

Scripting name	`selectionHighlight`
Data type	Color
Flags	expert

Track Margin

The amount of room on either side of the slider track. May need adjusting of default font is changed.

Scripting name	`trackMargin`
Data type	int
Flags	expert

High Density Color

The color used to indicate high data density.

Scripting name	`highDensityColor`
Data type	Color

Date Style

The style to display dates in. For international support.

Scripting name	`dateStyle`
Data type	int
Flags	expert
Values	**0** Auto **1** MDY **2** DMY **3** YMD

Time Style

The style to display times of day. For international support.

Scripting name	`timeStyle`
Data type	int
Flags	expert
Values	**15** Auto **16** 12 HR **17** 24 HR

Styles

Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Startup Mode

Controls whether or not this date range automatically assigns itself a starting range based on the current time

Scripting name	`startupMode`
Data type	int
Values	**0** None **1** Automatic

Startup Range

If startup mode is Automatic, this will be the starting range of time available for selection.

© 2011 Inductive Automation

Scripting name `startupRange`
Data type String

Startup Selection — If startup mode is Automatic, this will be the starting selected range.

Scripting name `startupSelection`
Data type String

Max Selection — The maximum size of the selected date range

Scripting name `maxSelectionSize`
Data type String

Tick Density — This is multiplied by the width to determine the current ideal tick unit.

Scripting name `tickDensity`
Data type float
Flags expert

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Enabled — If disabled, a component cannot be used.

Scripting name `enabled`
Data type boolean

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name `opaque`
Data type boolean

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values
- `0` Default
- `1` Crosshair
- `2` Text
- `3` Wait
- `12` Hand
- `13` Move
- `4` SW Resize
- `5` SE Resize
- `6` NW Resize
- `7` NE Resize
- `8` N Resize
- `9` S Resize

© 2011 Inductive Automation

10 W Resize
11 E Resize

Data

Start Date — The starting date of the currently selected range.

Scripting name	`startDate`
Data type	Date
Flags	bindable

End Date — The ending date of the currently selected range.

Scripting name	`endDate`
Data type	Date
Flags	bindable

Data Density — A dataset that is used to calculate a histogram of data density

Scripting name	`densityData`
Data type	Dataset

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

The ending date of the available outer range.

Outer Range End — Data

Scripting name	`outerRangeEndDate`
Data type	Date
Flags	bindable | expert

The starting date of the available outer range.

Outer Range Start — Data

Scripting name	`outerRangeStartDate`
Data type	Date
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

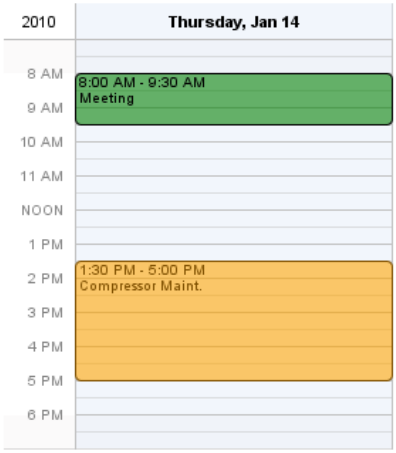
- mouse
- mouseMotion
- propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.6.4 Day View



Description

This component displays a timeline for a single day, similar to what you might find in a personal planner/organizer. By filling in the *Calendar Events* dataset property, the component will display events that occur on this day. Each event can have custom text and a custom display color associated with it. The format of the dataset requires 4 columns, as demonstrated by the following dataset:

StartDate (Date)	**EndDate** (Date)	**DisplayColor** (String)	**Display** (String)
2010-01-10 8:00:00	2010-01-10 9:30:00	color(0,180,0)	Meeting
2010-01-10 13:30:00	2010-01-10 17:00:00	orange	Compressor Maint.

Properties

Appearance

Working Start Hour — The start hour of a working day

Scripting name	`workingStartHour`
Data type	int
Flags	bindable

Working End Hour — The end hour of a working day

Scripting name	`workingEndHour`
Data type	int
Flags	bindable

24 Hour Format — Whether or not to show 24 hour or 12 hour format

Scripting name	`twentyFourHour`
Data type	boolean
Flags	bindable

Zoom — Zooms into the specified zoom time-range

Scripting name	`autoZoom`
Data type	boolean
Flags	bindable

Zoomed Start Hour — The start hour zoomed in

Scripting name	`autoZoomStartHour`
Data type	int

© 2011 Inductive Automation

	Flags	bindable
Zoomed End Hour	The end hour zoomed in	
	Scripting name	`autoZoomEndHour`
	Data type	int
	Flags	bindable
Grid marks	Set the amount of grid lines	
	Scripting name	`gridMarks`
	Data type	int
	Flags	bindable
Week Day Foreground Color	The color of the week day's text.	
	Scripting name	`weekDaysForeground`
	Data type	Color
	Flags	bindable
Week Day Background Color	The color of the week day's background	
	Scripting name	`weekDaysBackground`
	Data type	Color
	Flags	bindable
Calendar Background Color	The color of the calendar's background.	
	Scripting name	`calendarBackground`
	Data type	Color
	Flags	bindable
Day Outline Color	The color of the day's outline	
	Scripting name	`boxOutline`
	Data type	Color
	Flags	bindable
Today's Background Color	The color of the today's background	
	Scripting name	`todayBackground`
	Data type	Color
	Flags	bindable
Hover Background Color	The background color of the hovered time	
	Scripting name	`hoverBackground`
	Data type	Color
	Flags	bindable
Hour Foreground Color	The foreground color for hours in a day	
	Scripting name	`hourForeground`
	Data type	Color
	Flags	bindable
Non-Working Hours Background Color	The background color for the non-working hours of the day	
	Scripting name	`nonWorkingHourBackground`
	Data type	Color
	Flags	bindable
Styles	Contains the component's styles	
	Scripting name	`styles`
	Data type	Dataset
	Flags	bindable | expert

Common

© 2011 Inductive Automation

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default **1** Crosshair **2** Text **3** Wait **12** Hand **13** Move **4** SW Resize **5** SE Resize **6** NW Resize **7** NE Resize **8** N Resize **9** S Resize **10** W Resize **11** E Resize

Data

Year — Set the calendar's year

Scripting name	`year`
Data type	int
Flags	bindable

Month — Set the calendar's month

Scripting name	`month`
Data type	int
Flags	bindable

Day — Set the calendar's day

Scripting name	`day`
Data type	int
Flags	bindable

Calendar events — Contains the calendar events

Scripting name	`events`
Data type	Dataset
Flags	bindable

© 2011 Inductive Automation

Hovered Time	The calendar's hovered time
	Scripting name `hoveredTime` Data type String Flags bindable
Selected Event	The calendar's selected event
	Scripting name `selectedEvent` Data type int Flags bindable
Hovered Event	The calendar's hovered event
	Scripting name `hoveredEvent` Data type int Flags bindable
Data Quality	The data quality code for any tag bindings on this component.
	Scripting name `dataQuality` Data type int Flags bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

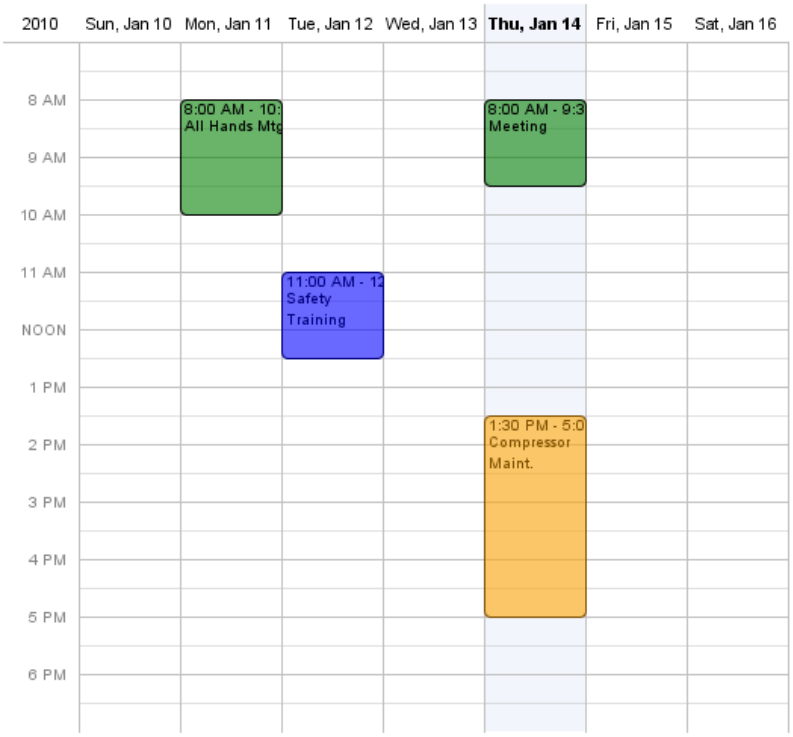
- mouse
- mouseMotion
- propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.6.5 Week View



Description

Displays a full week's worth of events on a calendar. Configuration is achieved by populating the *Calendar Events* dataset. See the Day View for details.

Properties

Appearance

Working Start Hour — The start hour of a working day

Scripting name	`workingStartHour`
Data type	int
Flags	bindable

Working End Hour — The end hour of a working day

Scripting name	`workingEndHour`
Data type	int
Flags	bindable

24 Hour Format — Whether or not to show 24 hour or 12 hour format

Scripting name	`twentyFourHour`
Data type	boolean
Flags	bindable

Zoom — Zooms into the specified zoom time-range

Scripting name	`autoZoom`
Data type	boolean
Flags	bindable

Zoomed Start Hour — The start hour zoomed in

Scripting name	`autoZoomStartHour`

© 2011 Inductive Automation

	Data type	int
	Flags	bindable
Zoomed End Hour	The end hour zoomed in	
	Scripting name	autoZoomEndHour
	Data type	int
	Flags	bindable
Grid marks	Set the amount of grid lines	
	Scripting name	gridMarks
	Data type	int
	Flags	bindable
Week Day Foreground Color	The color of the week day's text.	
	Scripting name	weekDaysForeground
	Data type	Color
	Flags	bindable
Week Day Background Color	The color of the week day's background	
	Scripting name	weekDaysBackground
	Data type	Color
	Flags	bindable
Calendar Background Color	The color of the calendar's background.	
	Scripting name	calendarBackground
	Data type	Color
	Flags	bindable
Day Outline Color	The color of the day's outline	
	Scripting name	boxOutline
	Data type	Color
	Flags	bindable
Today's Background Color	The color of the today's background	
	Scripting name	todayBackground
	Data type	Color
	Flags	bindable
Selected Background Color	The color of the selected day's background	
	Scripting name	selectedBackground
	Data type	Color
	Flags	bindable
Hover Background Color	The background color of the hovered day and time	
	Scripting name	hoverBackground
	Data type	Color
	Flags	bindable
Hour Foreground Color	The foreground color for hours in a day	
	Scripting name	hourForeground
	Data type	Color
	Flags	bindable
Non-Working Hours Background Color	The background color for the non-working hours of the day	
	Scripting name	nonWorkingHourBackground
	Data type	Color
	Flags	bindable
Styles	Contains the component's styles	

© 2011 Inductive Automation

Scripting name styles
Data type Dataset
Flags bindable | expert

Common

Name — The name of this component.

Scripting name name
Data type String
Flags bindable

Visible — If disabled, the component will be hidden.

Scripting name visible
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name border
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name toolTipText
Data type String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name cursorCode
Data type int
Values
- 0 Default
- 1 Crosshair
- 2 Text
- 3 Wait
- 12 Hand
- 13 Move
- 4 SW Resize
- 5 SE Resize
- 6 NW Resize
- 7 NE Resize
- 8 N Resize
- 9 S Resize
- 10 W Resize
- 11 E Resize

Data

Year — Set the calendar's year

Scripting name year
Data type int
Flags bindable

Month — Set the calendar's month

Scripting name month
Data type int
Flags bindable

Day — Set the calendar's day

Scripting name day
Data type int
Flags bindable

© 2011 Inductive Automation

Calendar events — Contains the calendar events

Scripting name	`events`
Data type	Dataset
Flags	bindable

Selected Day — The calendar's selected day

Scripting name	`selectedDay`
Data type	String
Flags	bindable

Hovered Day — The calendar's hovered day

Scripting name	`hoveredDay`
Data type	String
Flags	bindable

Hovered Time — The calendar's hovered time

Scripting name	`hoveredTime`
Data type	String
Flags	bindable

Selected Event — The calendar's selected event

Scripting name	`selectedEvent`
Data type	int
Flags	bindable

Hovered Event — The calendar's hovered event

Scripting name	`hoveredEvent`
Data type	int
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.6.6 Month View

Description

Displays a month's worth of events on a calendar. Configuration is achieved by populating the *Calendar Events* dataset. See the Day View for details.

Properties

Appearance

Header Font — The font of the header's text.

Scripting name	`headerFont`
Data type	Font

Event Display Mode — Affects how events are displayed.Standard Mode: Displays each eventHighlight Mode: Highlights each day that contains events using the event highlight background color.

Scripting name	`displayMode`
Data type	int
Flags	bindable
Values	**1** Standard **2** Highlight

Event Highlight Background — The background color of a day with events. Used only in highlight mode.

Scripting name	`highlightBackground`
Data type	Color
Flags	bindable

Header Foreground Color — The color of the header's text.

Scripting name	`monthHeaderForeground`
Data type	Color
Flags	bindable

Header Background Color — The color of the header's background

Scripting name	`monthHeaderBackground`
Data type	Color

© 2011 Inductive Automation

	Flags	bindable
Week Day Font	The font of the week day's text.	
	Scripting name	`weekdayFont`
	Data type	Font
Week Day Foreground Color	The color of the week day's text.	
	Scripting name	`weekDaysForeground`
	Data type	Color
	Flags	bindable
Week Day Background Color	The color of the week day's background	
	Scripting name	`weekDaysBackground`
	Data type	Color
	Flags	bindable
Calendar Background Color	The color of the calendar's background.	
	Scripting name	`calendarBackground`
	Data type	Color
	Flags	bindable
Today's Background Color	The color of the today's background	
	Scripting name	`todayBackground`
	Data type	Color
	Flags	bindable
Selected Background Color	The color of the selected day's background	
	Scripting name	`selectedBackground`
	Data type	Color
	Flags	bindable
Hover Background Color	The background color of the hovered day	
	Scripting name	`hoverBackground`
	Data type	Color
	Flags	bindable
Day Outline Color	The color of the day's outline	
	Scripting name	`boxOutline`
	Data type	Color
	Flags	bindable
Day Font	The font for the number representing the day of the month.	
	Scripting name	`dayFont`
	Data type	Font
Day Foreground Color	The foreground color for days in this month	
	Scripting name	`dayOfMonthForeground`
	Data type	Color
	Flags	bindable
Day Other Foreground Color	The foreground color for days not in this month	
	Scripting name	`dayOfMonthOtherForeground`
	Data type	Color
	Flags	bindable
Event Font	The font for all calendar events.	
	Scripting name	`eventFont`
	Data type	Font

© 2011 Inductive Automation

Event Background Color — The background color of the selected event

Scripting name	`itemSelBackground`
Data type	Color
Flags	bindable

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default **1** Crosshair **2** Text **3** Wait **12** Hand **13** Move **4** SW Resize **5** SE Resize **6** NW Resize **7** NE Resize **8** N Resize **9** S Resize **10** W Resize **11** E Resize

Data

Month — Set the calendar's month

Scripting name	`month`
Data type	int
Flags	bindable

Year — Set the calendar's year

Scripting name	`year`
Data type	int

© 2011 Inductive Automation

Flags	bindable

Calendar events — Contains the calendar events

Scripting name	`events`
Data type	Dataset
Flags	bindable

Selected Day — The calendar's selected day

Scripting name	`selectedDay`
Data type	String
Flags	bindable

Hovered Day — The calendar's hovered day

Scripting name	`hoveredDay`
Data type	String
Flags	bindable

Selected Event — The calendar's selected event

Scripting name	`selectedEvent`
Data type	int
Flags	bindable

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

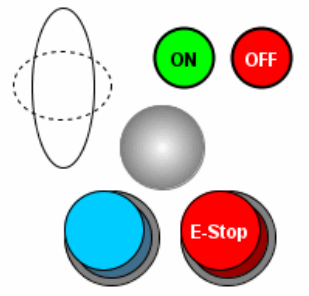
mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.7 Shapes

7.7.1 Circle



Description

The circle component displays circle or an oval. The oval has a variable width outline and can optionally be filled with a color, or a circular gradient. In addition, you can enter a string which will be

© 2011 Inductive Automation

displayed in the center of the circle. To make the oval a perfect circle, remember to hold down Shift while creating/resizing a component to constrain proportions. (See Keyboard Shortcuts)

You can add dynamic properties to circles, so they make great status indicators. The "buttons" in the screenshot are made up of three circles overlapping.

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Outline Color — The color of circle's outline.

Scripting name	`foreground`
Data type	Color
Flags	bindable

Fill Color — The fill color for the circle

Scripting name	`background`
Data type	Color
Flags	bindable

Gradient Color — The color to mix with the fill color to make the gradient.

Scripting name	`gradientColor`
Data type	Color
Flags	bindable

Label Color — The color of the text in the circle, if any.

Scripting name	`textColor`
Data type	Color

Fill? — Should the circle be filled with the fill color?

Scripting name	`fill`
Data type	boolean

Outline? — Should an outline be drawn around the circle?

Scripting name	`outline`
Data type	boolean

Label — You may display some text in the middle of your circle

Scripting name	`label`
Data type	String

Line Width — Set the width of the outline in pixels.

Scripting name	`lineWidth`
Data type	float
Flags	bindable

Line Style — The line style determines how the shape of the line looks

Scripting name	`lineStyle`
Data type	int
Flags	bindable
Values	`0` Plain
	`1` Dashed

Dash Pattern — Enter a string of comma-delimited numbers which indicate the stroke

© 2011 Inductive Automation

pattern for a dashed outline. For instance, "3,5" means three pixels on, five pixels off.

Scripting name `strokePattern`
Data type String

Gradient Type

The type gradient of gradient to use to fill the circle

Scripting name `gradientType`
Data type int
Values
0 None
1 Linear
2 Spherical

Gradient Length

This is multiplied by the radius to use to determine gradient length.

Scripting name `gradientLength`
Data type double
Flags expert

Gradient Angle

The starting angle for linear gradients

Scripting name `gradientAngle`
Data type int
Flags expert

Gradient Cyclic?

If true, the linear gradient will repeat

Scripting name `gradientRepeat`
Data type boolean
Flags expert

Rotation

The angle of rotation in degrees.

Scripting name `rotation`
Data type int

Anti Alias

Draw component using anti-aliasing?

Scripting name `antiAlias`
Data type boolean
Flags expert

Styles

Contains the component's styles

Scripting name `styles`
Data type Dataset
Flags bindable | expert

Common

Name

The name of this component.

Scripting name `name`
Data type String
Flags bindable

Visible

If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor

The mouse cursor to use when hovering over this component.

© 2011 Inductive Automation

Scripting name	`cursorCode`
Data type	int
Values	`0` Default `1` Crosshair `2` Text `3` Wait `12` Hand `13` Move `4` SW Resize `5` SE Resize `6` NW Resize `7` NE Resize `8` N Resize `9` S Resize `10` W Resize `11` E Resize

Data

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Deprecated

Old Fill Color — For backwards compatability only. Use "Fill Color"

Scripting name	`fillColor`
Data type	Color
Flags	expert

Old Outline Color — For backwards compatability only. Use "Outline Color"

Scripting name	`outlineColor`
Data type	Color
Flags	expert

Scripting

Events

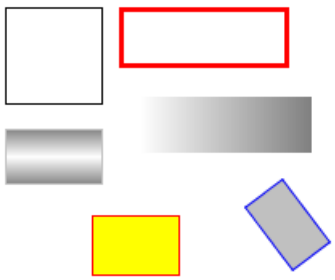
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.7.2 Rectangle



© 2011 Inductive Automation

Description

The rectangle component displays a square or a rectangle. Remember, if you want a square, hold down Shift while creating/resizing the rectangle. (see Keyboard Shortcuts) The rectangle component has control over what sides are shown, how thick they are, if the center is filled, gradient fill, etc.

Properties

Appearance

Rotation — The angle of rotation in degrees.

Scripting name	`rotation`
Data type	int

Foreground Color — The color of the sides of this rectangle.

Scripting name	`foreground`
Data type	Color
Flags	bindable

Background Color — The color of the fill of this rectangle.

Scripting name	`background`
Data type	Color
Flags	bindable

Draw Fill — If true, the rectangle will be filled with the background color or a gradient.

Scripting name	`drawFill`
Data type	boolean

North Edge Line Width — The line width of the north edge of the rectangle, if drawn.

Scripting name	`northLineWidth`
Data type	int

East Edge Line Width — The line width of the east edge of the rectangle, if drawn.

Scripting name	`eastLineWidth`
Data type	int

South Edge Line Width — The line width of the south edge of the rectangle, if drawn.

Scripting name	`southLineWidth`
Data type	int

West Edge Line Width — The line width of the west edge of the rectangle, if drawn.

Scripting name	`westLineWidth`
Data type	int

Rounding Radius — If non-zero, and if all edges have the same width, this rectangle will be rounded with the given radius.

Scripting name	`roundingRadius`
Data type	float

Gradient Type — The gradient type

Scripting name	`gradientType`
Data type	int
Values	0 Off 1 North-South 2 East-West

Gradient Style — The gradient style

Scripting name	`gradientStyle`

© 2011 Inductive Automation

Data type int
Values
1 Round
0 Slope

Gradient Color — The color to mix with the background color to make the gradient.

Scripting name `gradientColor`
Data type Color
Flags bindable

Styles — Contains the component's styles

Scripting name `styles`
Data type Dataset
Flags bindable | expert

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values
0 Default
1 Crosshair
2 Text
3 Wait
12 Hand
13 Move
4 SW Resize
5 SE Resize
6 NW Resize
7 NE Resize
8 N Resize
9 S Resize
10 W Resize
11 E Resize

Data

Data Quality — The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Layout

Override Size? — Check this to manually set the size of the rectangle. Usefull for rotating

Scripting name `overrideSize`
Data type boolean

© 2011 Inductive Automation

Overridden Width

Scripting name	`overrideWidth`
Data type	int

Overridden Height

Scripting name	`overrideHeight`
Data type	int

Scripting

Events

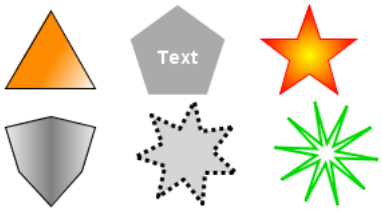
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.7.3 Polygon



Description

The polygon is a versatile shape displaying a regular polygon of 3 or more vertices at any degree of rotation. As an added mode, the polygon can become a "star", which means an additional vertex is added between each normal vertex, but at a different distance from the center of the shape. Combine this with line styles, widths, colors, gradient fills, and the ability to display text, and you get a simple but powerful shape component.

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Outline Color — The color of polygon/star outline.

Scripting name	`foreground`
Data type	Color
Flags	bindable

Fill Color — The fill color for the polygon/star

Scripting name	`background`
Data type	Color
Flags	bindable

Gradient Color — The color to mix with the fill color to make the gradient.

© 2011 Inductive Automation

Scripting name	`gradientColor`
Data type	Color
Flags	bindable

Label Color

The color of the text in the polygon/star, if any.

Scripting name	`textColor`
Data type	Color

Vertices

The number of vertices (corners) for the polygon/star

Scripting name	`vertices`
Data type	int
Flags	bindable

Star?

If true, the shape will be a star. Otherwise, the shape will be a polygon

Scripting name	`star`
Data type	boolean
Flags	bindable

Spoke Ratio

The ratio of the star's inner spoke vertices' radii compared to the outer ones

Scripting name	`spokeRatio`
Data type	double

Fill?

If true, the polygon/star will be filled in

Scripting name	`fill`
Data type	boolean

Outline?

If true, an outline will be drawn around the polygon/star

Scripting name	`outline`
Data type	boolean

Label

You may display some text in the middle of your polygon/star

Scripting name	`label`
Data type	String

Line Width

Set the width of the outline in pixels.

Scripting name	`lineWidth`
Data type	float
Flags	bindable

Line Style

The line style determines how the shape of the line looks

Scripting name	`lineStyle`
Data type	int
Flags	bindable
Values	**0** Plain
	1 Dashed

Dash Pattern

Enter a string of comma-delimited numbers which indicate the stroke pattern for a dashed outline. For instance, "3,5" means three pixels on, five pixels off.

Scripting name	`strokePattern`
Data type	String

Gradient Type

The type gradient of gradient to use to fill the polygon/star

Scripting name	`gradientType`
Data type	int
Values	**0** None
	1 Linear
	2 Spherical

© 2011 Inductive Automation

Gradient Length — This is multiplied by the radius to use to determine gradient length.

Scripting name	`gradientLength`
Data type	double
Flags	expert

Gradient Angle — The starting angle for linear gradients

Scripting name	`gradientAngle`
Data type	int
Flags	expert

Gradient Cyclic? — If true, the linear gradient will repeat

Scripting name	`gradientRepeat`
Data type	boolean
Flags	expert

Rotation — The angle of rotation in degrees.

Scripting name	`rotation`
Data type	int

Antialias — Draw with antialias on? Makes shape smoother

Scripting name	`antiAlias`
Data type	boolean
Flags	expert

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	`0` Default `1` Crosshair `2` Text `3` Wait `12` Hand `13` Move

© 2011 Inductive Automation

4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

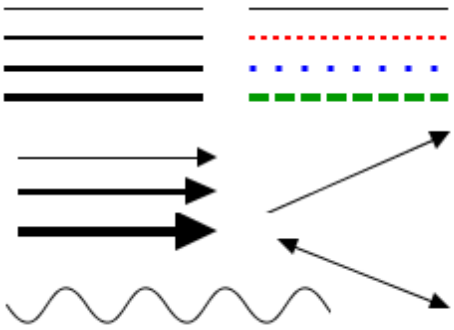
The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- propertyChange

Scripting Functions

This component has no special scripting functions.

7.7.4 Line



Lines come in many styles

Description

The line component displays a straight line. It can run north-south, east-west, or diagonally. You can add arrows to either side. The line can be dashed using any pattern you want. You can even draw the line like a sinusoidal wave!

Properties

Appearance

Anti Alias — Draw component using anti-aliasing?

Scripting name	`antiAlias`
Data type	boolean
Flags	expert

Color — Set the color of the line.

© 2011 Inductive Automation

Scripting name	foreground
Data type	Color
Flags	bindable

Line Width

Set the width of the line in pixels.

Scripting name	lineWidth
Data type	int
Flags	bindable

Line Mode

The line mode determines where in the rectangle the line is drawn.

Scripting name	lineMode
Data type	int
Flags	bindable
Values	0 Horizontal/Vertical 1 Uphill (Left-Right) 2 Dow nhill (Left-Right)

Line Style

The line style determines how the shape of the line looks

Scripting name	lineStyle
Data type	int
Flags	bindable
Values	0 Plain 1 Dashed 2 Sinusoidal 3 Sinusoidal-Dashed 4 Loop 5 Loop-Dashed

Dash Pattern

Enter a string of comma-delimited numbers which indicate the stroke pattern for a dashed line. For instance, "3,5" means three pixels on, five pixels off.

Scripting name	strokePattern
Data type	String

Sine Length

Sets the 'wavelength' of the sine wave to be drawn

Scripting name	sineLength
Data type	int

Sine Height

Sets the 'amplitude' of the sine wave to be drawn

Scripting name	sineHeight
Data type	int

Left Arrow

Draw an arrow head on the left/top of the line?

Scripting name	leftArrow
Data type	boolean

Right Arrow

Draw an arrow head on the right/bottom of the line?

Scripting name	rightArrow
Data type	boolean

Left Arrow Size

The size of the left arrow, if present

Scripting name	leftArrowSize
Data type	int

Right Arrow Size

The size of the right arrow, if present

Scripting name	rightArrowSize
Data type	int

Styles

Contains the component's styles

© 2011 Inductive Automation

Scripting name `styles`
Data type Dataset
Flags bindable | expert

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Data Quality — The data quality code for any tag bindings on this component.

Scripting name `dataQuality`
Data type int
Flags bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

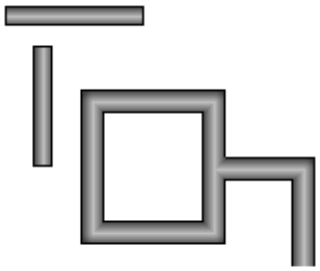
- mouse
- mouseMotion
- propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.7.5 Pipe Segment

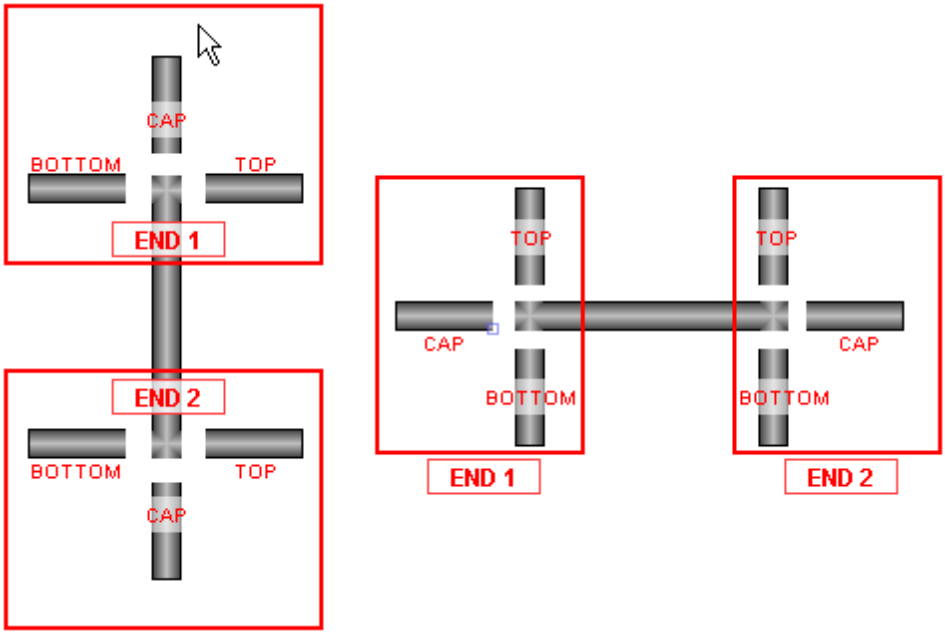


Pipe Segments, apart and joined together

Description

The pipe segment component displays a quasi-3D pipe. In its basic form it looks very much like a rectangle with a round gradient. The difference comes in its advanced rendering of its edges and endcaps. You can configure each pipe segment's end to mate perfectly with another pipe segment butted up against it perpendicularly. The result looks like a pipe welded together in a 90° corner.

The control of the pipe's ends can be a bit confusing to a new user. It is done via 6 booleans - three per 'end'. End 1 is the top/left end, and End 2 is the bottom/right end. You turn off each boolean if there will be another pipe butted up against that side. The following diagram should make the naming conventions more clear:



Properties

Appearance

Center Fill	The center of the fill gradient
Scripting name	`mainColor`

© 2011 Inductive Automation

Data type	Color
Flags	bindable

Edge Fill — The edge of the fill gradient.

Scripting name	`secondaryColor`
Data type	Color
Flags	bindable

Outline Color — The color of the outline border

Scripting name	`outlineColor`
Data type	Color
Flags	bindable

End 1 Top? — Draw the border at end #1's top?

Scripting name	`end1Top`
Data type	boolean

End 1 Cap? — Draw the border at end #1's cap?

Scripting name	`end1Cap`
Data type	boolean

End 1 Bottom? — Draw the border at end #1's bottom?

Scripting name	`end1Bottom`
Data type	boolean

End 2 Top? — Draw the border at end #2's top?

Scripting name	`end2Top`
Data type	boolean

End 2 Cap? — Draw the border at end #2's cap?

Scripting name	`end2Cap`
Data type	boolean

End 2 Bottom? — Draw the border at end #2's bottom?

Scripting name	`end2Bottom`
Data type	boolean

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of

© 2011 Inductive Automation

this component.

Scripting name	`toolTipText`
Data type	String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	`0` Default `1` Crosshair `2` Text `3` Wait `12` Hand `13` Move `4` SW Resize `5` SE Resize `6` NW Resize `7` NE Resize `8` N Resize `9` S Resize `10` W Resize `11` E Resize

Data

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

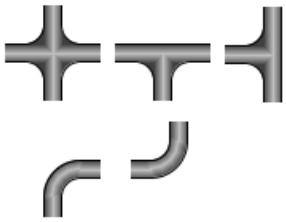
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.7.6 Pipe Joint



Pipe Joint components

Description

The pipe joint displays a fancy joint component two join two pipe segments together. By turning off the cardinal directions, this will display a 2,3, or 4-pipe union. This component is optional, as pipes can butt up against each other and look joined..

© 2011 Inductive Automation

Properties

Appearance

Center Fill — The center of the fill gradient

Scripting name	`mainColor`
Data type	Color
Flags	bindable

Edge Fill — The edge of the fill gradient.

Scripting name	`secondaryColor`
Data type	Color
Flags	bindable

Outline Color — The color of the outline border

Scripting name	`outlineColor`
Data type	Color
Flags	bindable

Top? — Joint has an outlet at the top?

Scripting name	`top`
Data type	boolean

Right? — Joint has an outlet at the right?

Scripting name	`right`
Data type	boolean

Bottom? — Joint has an outlet at the bottom?

Scripting name	`bottom`
Data type	boolean

Left? — Joint has an outlet at the left?

Scripting name	`left`
Data type	boolean

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Visible — If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

© 2011 Inductive Automation

Scripting name	`toolTipText`
Data type	String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	`0` Default `1` Crosshair `2` Text `3` Wait `12` Hand `13` Move `4` SW Resize `5` SE Resize `6` NW Resize `7` NE Resize `8` N Resize `9` S Resize `10` W Resize `11` E Resize

Data

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

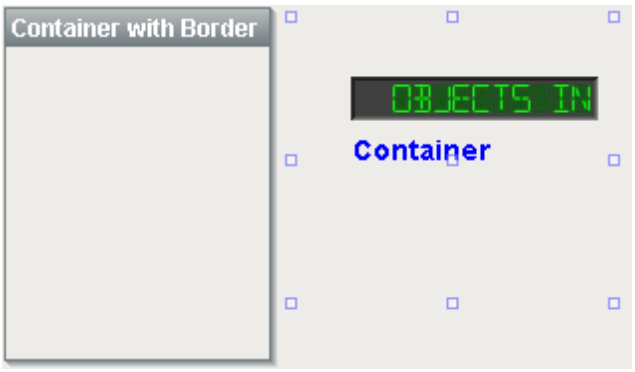
mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.8 Misc

7.8.1 Container



Description

The container is a very important component. All components are always inside of a container,

except for the special "Root Container" of each window (see Anatomy of a Window). A container is different than normal components in that it can contain other components, including other containers. Uses for containers include:

Organization. Containers can be used to group components together. These components can then easily be moved, copied, or deleted as a group. Furthermore, they will all be organized inside of their parent container in the project navigation tree, which makes them easier to find.
Re-usability. Containers allow a unique opportunity to create a complex component that is made up of multiple other components. The Container's ability to have dynamic properties aids this greatly. For instance, if you wanted to make your own custom HOA control, you can put three buttons inside of a container and configure them to all use a 'status' property that you add to their parent Container. Now you have built an HOA control that can be re-used and treated like its own component. The possibilities here are endless. Create a date range control that generates an SQL WHERE clause that can be used to control Charts and Tables. Create a label/button control that can be used to display datapoints, and pop up a parameterized window that displays meta-data (engineering units, physical location, notes, etc) about that datapoint. Creating re-usable controls with Containers containing multiple components is the key to rapid application development.
Layout. Containers are a great way to improve window aesthetics through borders and layout options.

Grouping

A container can be set as a "group" by right-clicking on it and choosing "Group Container". This will make the container act like a single component - you won't be able to select its children by clicking on them. This can help make window design easier, as you'll always pick the container by clicking anywhere inside it. You can still get to the individual sub-components by choosing them in the project navigation tree. You can un-group a container at any time by right clicking on it and choosing "Ungroup Container".

See also:
Component Layout
Custom Palettes

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Background Color — Set the color of the background

Scripting name	`background`
Data type	Color
Flags	bindable

Texture — Background texture image for this container.

Scripting name	`texturePath`
Data type	String

Styles — Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

© 2011 Inductive Automation

Optimized for tiling?

If true, this container's children should never overlap, and you'll get better painting performance.

Scripting name	`optimizedDrawingEnabled`
Data type	boolean
Flags	expert

Common

Name

The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled

If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible

If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border

The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text

The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Opaque

If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name	`opaque`
Data type	boolean

Cursor

The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	**0** Default
	1 Crosshair
	2 Text
	3 Wait
	12 Hand
	13 Move
	4 SW Resize
	5 SE Resize
	6 NW Resize
	7 NE Resize
	8 N Resize
	9 S Resize
	10 W Resize
	11 E Resize

Data

Data Quality

The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

© 2011 Inductive Automation

Scripting

Events

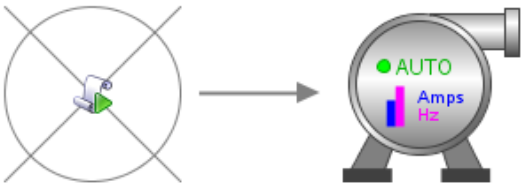
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.8.2 Paintable Canvas



A paintable canvas in Design and Preview modes

Description

The Paintable Canvas component is a component that can be custom "painted" using Jython scripting. By responding to the component's `repaint` event, a designer can use Java2D to draw anything within the component's bounds. Whenever any dynamic properties on the component change, the component is re-painted automatically, making it possible to create dynamic, vector-drawn components that can represent anything.

This component is an **advanced component** for those who are very comfortable using scripting. It is *not* user-friendly. The upside is that it is extraordinarily powerful, as your imagination is the only limit with what this component can be.

When you first drop a Paintable Canvas onto a window, you'll notice that it looks like a placeholder. If you switch the Designer into *preview* mode, you'll see an icon of a pump displayed. The pump is an example that comes pre-loaded into the Paintable Canvas. By editing the component's event scripts, you can dissect how the pump was drawn. You will notice that the script uses Java2D. You can read more about Java2D here http://java.sun.com/docs/books/tutorial/2d/index.html. You will notice that as you resize the pump, it scales beautifully in preview mode. Java2D is a vector drawing library, enabling you to create components that scale very gracefully.

Tips:

Don't forget that you can add dynamic properties to this component, and use the styles feature.
Use the values of dynamic properties in your repaint code to create a dynamic component. The component will repaint automatically when these values change.
You can create an interactive component by responding to mouse and keyboard events
You can store your custom components on a custom palette and use them like standard components.

See also:
Event Types - repaint

© 2011 Inductive Automation

Properties

Appearance

Font
Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color
The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color
The background color of the component.

Scripting name	`background`
Data type	Color

Styles
Contains the component's styles

Scripting name	`styles`
Data type	Dataset
Flags	bindable | expert

Behavior

Focusable
If the component is focusable, it will recieve keyboard input and can detect if it is the focus owner.

Scripting name	`focusable`
Data type	boolean
Flags	expert

Common

Name
The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

Enabled
If disabled, a component cannot be used.

Scripting name	`enabled`
Data type	boolean

Visible
If disabled, the component will be hidden.

Scripting name	`visible`
Data type	boolean

Border
The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name	`border`
Data type	Border

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name	`toolTipText`
Data type	String

Cursor
The mouse cursor to use when hovering over this component.

Scripting name	`cursorCode`
Data type	int
Values	`0` Default
	`1` Crosshair

© 2011 Inductive Automation

2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Data

Data Quality — The data quality code for any tag bindings on this component.

Scripting name	`dataQuality`
Data type	int
Flags	bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- focus
- paint
- propertyChange
- key

Scripting Functions

This component has no special scripting functions.

7.8.3 Sound Player

Description

The Sound Player component is an invisible component that facilitates audio playback in the client. Each Sound Player component has one sound clip associated with it, and will play that clip on demand. There is a built in triggering system, as well as facilities to loop the sound while the trigger is set. Note that the sound clip needs to be a *.wav file, and that the clip becomes embedded within the window that the sound player is on - clients do *not* need access to a shared *.wav file.

Properties

Behavior

Play Mode — The Play Mode determines whether the sound is played automatically on trigger or manually

Scripting name	`playMode`
Data type	int
Values	0 Manual
	1 On Trigger

Loop Mode — The Loop Mode determines how many times the sound is played when triggered.

Scripting name	`loopMode`
Data type	int

© 2011 Inductive Automation

Values
0 Play Once
1 Loop Forever
2 Loop N Times

Loop Count
If Loop Mode is "Loop N Times", this is the "N".
Scripting name loopCount
Data type int

Volume
The volume to use for playback (from 0.0 to 1.0).
Scripting name volume
Data type double

Mute
If true, the clip will be muted during playback.
Scripting name mute
Data type boolean

Common

Name
The name of this component.
Scripting name name
Data type String
Flags bindable

Mouseover Text
The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name toolTipText
Data type String

Data

Trigger
The clip will be played when the trigger is true, if Play Mode is "ON_TRIGGER"
Scripting name trigger
Data type boolean
Flags bindable

Sound Data
The clip that this component will play.
Scripting name soundData
Data type byte[]

Data Quality
The data quality code for any tag bindings on this component.
Scripting name dataQuality
Data type int
Flags bindable | expert

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.8.4 Timer

Description

The timer button is an invisible button that can be used to create repeated events in a window. This is often used for animations or repetitive scripts within a window. When running, the timer's *Value* property is incremented by the *Step By* value, until the value tis the *Bound*, at which point it repeats. It is often useful to bind other values to a timer's *Value* property.

For instance, if you set the timer's *Bound* property to 360, and bind an object's rotation to the *Value* property, the object will spin in a circle when the timer is running.

Or, suppose that you have images that make up frames of animation. Name your images: "`Frame0.png`", "`Frame1.png`", "`Frame2.png`". Set the timer's *Bound* to be 3, Then bind the *image path* of an image component to the following expression:

```
"Frame" + {Root Container.Timer.value} + ".png"
```

How fast the timer counts is up to the *Delay* property, which is the time between counts in milliseconds.

Want to run a script every time the timer counts? First, make sure you don't actually want to write a project Timer Script, which will run on some interval whenever the application is running. In contrast, a script that works via a Timer *component* will only run while the window that contains the Timer is open, and the Timer is running. The way to do this is to attach an event script to the `actionPerformed` event.

Properties

Behavior

Delay (ms)
: The delay in milliseconds between timer events.

Scripting name	`delay`
Data type	int
Flags	bindable

Initial Delay (ms)
: The delay in milliseconds before the first event when running is set to true.

Scripting name	`initialDelay`
Data type	int
Flags	bindable

Running?
: Determines whether or not the timer sends timer events.

Scripting name	`running`
Data type	boolean
Flags	bindable

Common

Name
: The name of this component.

Scripting name	`name`
Data type	String

Data

Value
: The current value of this timer, for use as a counter.At each iteration, this value will be set to ((value + step) MOD bound)

Scripting name	`value`
Data type	int

© 2011 Inductive Automation

Flags bindable

Step by
The amount added to the value each time this timer fires for use as a counter. (should be positive)
Scripting name `step`
Data type int

Bound
The value is always guaranteed to be less than this upper bound.
Scripting name `max`
Data type int

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

action
propertyChange

Scripting Functions

This component has no special scripting functions.

7.8.5 Signal Generator

Description

The signal generator is similar to the Timer component, but its value isn't simply a counter. Instead, you can choose from a variety of familiar 'signals'. You configure the frequency by setting the *Period* property, which is in milliseconds. You configure the resolution by setting the *Values/Period* property.

For example, if you choose a sine wave signal with a period of 2000 milliseconds and 10 values/period, your sine wave will have a frequency of 0.5 Hz, and its value will change 10 times every 2 seconds.

Properties

Behavior

Signal Type
The signal type (shape) of the signal value
Scripting name `signalType`
Data type int
Values
0 Sine
2 Triangular
1 Ramp
3 Square
4 Random

Running?
Determines whether or not the signal is being generated.
Scripting name `running`
Data type boolean
Flags bindable

Period
The period of the signal in milliseconds
Scripting name `period`
Data type int

Values/Period
The number of value changes per period
Scripting name `valuesPerPeriod`
Data type int

© 2011 Inductive Automation

Common

Name — The name of this component.

Scripting name: `name`
Data type: String

Data

Value — The current value of this signal generator.

Scripting name: `value`
Data type: double
Flags: bindable

Upper Bound — The upper bound of the signal value.

Scripting name: `upper`
Data type: double

Lower Bound — The lower bound of the signal value.

Scripting name: `lower`
Data type: double

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

action
propertyChange

Scripting Functions

This component has no special scripting functions.

7.9 Reporting

7.9.1 Report Viewer

Description

This component is the heart of the Reporting Module. The customizer for this component is the Report Designer. See the Reporting section for more about creating dynamic reports.

Properties

Appearance

Font — Font of text of this component

Scripting name: `font`
Data type: Font

Foreground Color — The foreground color of the component.

Scripting name: `foreground`
Data type: Color

Background Color — The background color of the component.

Scripting name: `background`
Data type: Color

Zoom Factor — Sets the zoom factor for the report viewer

Scripting name: `zoomFactor`

© 2011 Inductive Automation

Data type float
Flags bindable

Behavior

Suggested Filename — The filename that will come up by default when the user saves the report to disk.

Scripting name `suggestedFilename`
Data type String

Print Mode — Sets the printing mode. Vector is fast and high-quality for printers that support it, but Raster mode can help the spool size with older printers.

Scripting name `printingMode`
Data type int
Flags expert
Values `0` Vector
`1` Raster

Raster DPI — If the mode is raster, this is the DPI that will be used.

Scripting name `printingDPI`
Data type int
Flags expert

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Enabled — If disabled, a component cannot be used.

Scripting name `enabled`
Data type boolean

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.

Scripting name `opaque`
Data type boolean

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

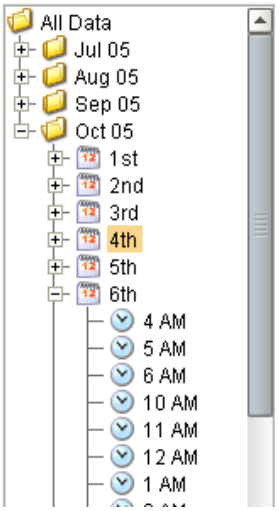
- mouse
- mouseMotion
- propertyChange

© 2011 Inductive Automation

Scripting Functions

This component has no special scripting functions.

7.9.2 Row Selector



Description

The row selector is a component that acts like a visual filter for datasets. It takes one dataset, chops it up into various ranges based on its configuration, and lets the user choose the splices. Then it creates a virtual dataset that only contains the rows that match the selected splices.

The most common way to splice the data is time. You could feed the row selector an input dataset that represents a large time range, and have it break it up by Month, Day, and then Shift, for example. Then you could power a report with the output dataset, and that would let the user dynamically create reports for any time range via an intuitive interface.

To configure the row selector, first you set up the appropriate bindings for its input dataset. Then you use its Customizer to alter the levels that it uses to break up the data. In the customizer, you add various filters that act upon columns in the input dataset, sorting them by various criteria. For example, you could choose a date column, and have it break that up by quarter. Then below that, you could have it use a discrete filter on a product code. This would let the user choose quarterly results for each product. Each level of filter you create in the customizer becomes a level in the selection hierarchy. Note that the output data is completely unchanged other than the fact that rows that don't match the current user selection aren't present.

This component is very handy for driving the Report Viewer, Table, and Classic Chart components, among others.

Properties

Appearance

Font	Font of text of this component
	Scripting name `font`
	Data type Font
Foreground Color	The foreground color of the component.
	Scripting name `foreground`

© 2011 Inductive Automation

Data type Color

Background Color — The background color of the component.
Scripting name background
Data type Color

All Data Node Text — Text for the 'All Data' node, if it is displayed
Scripting name allDataNodeText
Data type String
Flags expert

Unknown Node Text — Text for any 'Unknown' nodes (nodes where the data didn't match filter)
Scripting name unknownNodeText
Data type String
Flags expert

Unknown Node Icon — Icon for any 'Unknown' nodes (nodes where data didn't match filter)
Scripting name unknownIconPath
Data type String
Flags expert

Behavior

Show All Data Node — Should the 'All Data' (root) node be shown or hidden?
Scripting name showAllDataNode
Data type boolean

Show Root Handles — Should root-level nodes have collapse handles?
Scripting name showRootHandles
Data type boolean
Flags expert

Show Node Size — If true, the number of rows in each node will be shown
Scripting name showNodeSize
Data type boolean
Flags expert

Common

Name — The name of this component.
Scripting name name
Data type String
Flags bindable

Visible — If disabled, the component will be hidden.
Scripting name visible
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name border
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name toolTipText
Data type String

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.
Scripting name opaque

© 2011 Inductive Automation

Data type boolean

Cursor
The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values
`0` Default
`1` Crosshair
`2` Text
`3` Wait
`12` Hand
`13` Move
`4` SW Resize
`5` SE Resize
`6` NW Resize
`7` NE Resize
`8` N Resize
`9` S Resize
`10` W Resize
`11` E Resize

Data

Data In
The input of the row selection tree. The filter tree changes based on this DataSet.

Scripting name `dataIn`
Data type Dataset

Data Out
The output of the row selection tree. Changes based on user selection in the filter tree.

Scripting name `dataOut`
Data type Dataset
Flags bindable

Uncategorized

Properties Loading
The number of properties currently being loaded

Scripting name `propertiesLoading`
Data type int
Flags bindable | read-only

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

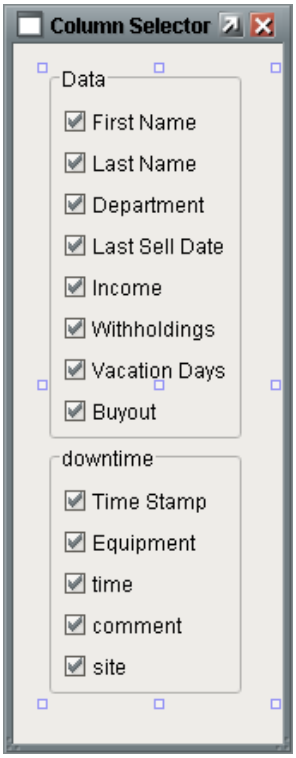
mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

7.9.3 Column Selector



Description

The column selector component is conceptually similar to the Row Selector, except that instead of filtering rows, it filters columns from its output dataset. Each column from the input dataset is shown as a checkbox. As the user checks and un-checks columns, the output dataset has those columns added or removed. This is very handy for driving the Table and Classic Chart components.

Properties

Appearance

Font	Font of text of this component Scripting name `font` Data type Font
Foreground Color	The foreground color of the component. Scripting name `foreground` Data type Color
Background Color	The background color of the component. Scripting name `background` Data type Color
Normalize Widths	If true, all checkboxes will be assigned the same width, which causes them to line up in columns Scripting name `normalizeWidths` Data type boolean Flags expert
Horizontal Gap	The horizontal gap between checkboxes or grouping panels

© 2011 Inductive Automation

Scripting name `hGap`
Data type int
Flags expert

Vertical Gap — The vertical gap between checkboxes and grouping panels

Scripting name `vGap`
Data type int
Flags expert

Behavior

Group by Dataset — If true, checkboxes will be grouped by their dataset. Otherwise, checkboxes will be arranged flat.

Scripting name `grouping`
Data type boolean

Alphabetize — If true, checkboxes will be ordered alphabetically by their text.

Scripting name `alphabetize`
Data type boolean

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize
11	E Resize

Scripting

© 2011 Inductive Automation

Events

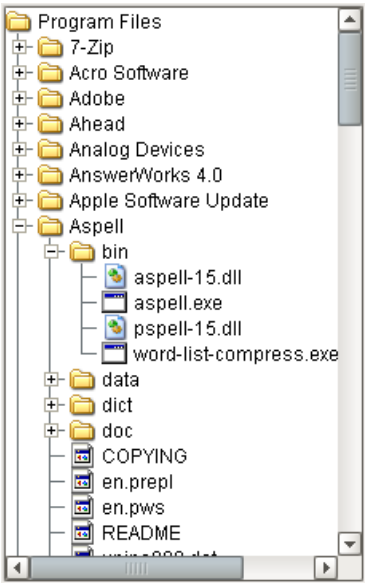
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.9.4 File Explorer



Description

The File Explorer component displays a filesystem tree to the user. It can be rooted at any folder, even network folders. It can also filter the types of files that are displayed by their file extension (For example, "`pdf`"). The path to the file that the user selects in the tree is exposed in the bindable property *Selected Path*.

This component is typically used in conjuction with the PDF Viewer component, in order to create a PDF viewing window. This is very useful for viewing things like maintenance manuals from within your project. To use this component to drive a PDF Viewer component, follow these steps:

1. Bind the PDF Viewer's *Filename* property to the File Explorer's *Selected Path* property
2. Set the File Explorer's *File extension filter* to "`pdf`"
3. Set the File Explorer's *Root Directory* to a network folder that has your maintenance manuals in it. (Use a network folder so that all clients will be able to access the manuals).

Properties

Appearance

Font	Font of text of this component
Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name `foreground`
Data type Color

Background Color — The background color of the component.

Scripting name `background`
Data type Color

Behavior

File extension filter — Semi-colon separated list of extensions to filter out files, such as pdf or txt. Example "pdf;html;txt" shows pdf, html and text documents.

Scripting name `fileFilter`
Data type String

Root Directory — A directory to act as the root of the file explorer.

Scripting name `rootDir`
Data type String

Common

Name — The name of this component.

Scripting name `name`
Data type String
Flags bindable

Enabled — If disabled, a component cannot be used.

Scripting name `enabled`
Data type boolean

Visible — If disabled, the component will be hidden.

Scripting name `visible`
Data type boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.

Scripting name `border`
Data type Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.

Scripting name `toolTipText`
Data type String

Cursor — The mouse cursor to use when hovering over this component.

Scripting name `cursorCode`
Data type int
Values

0	Default
1	Crosshair
2	Text
3	Wait
12	Hand
13	Move
4	SW Resize
5	SE Resize
6	NW Resize
7	NE Resize
8	N Resize
9	S Resize
10	W Resize

© 2011 Inductive Automation

11 E Resize

Data

Selected Path	The selected file or folder's path.
Scripting name	`selectedPath`
Data type	String
Flags	bindable
Selected Path Is File	True if the selected path is a file, not a directory.
Scripting name	`selectedPathIsFile`
Data type	boolean
Flags	bindable | read-only

Scripting

Events

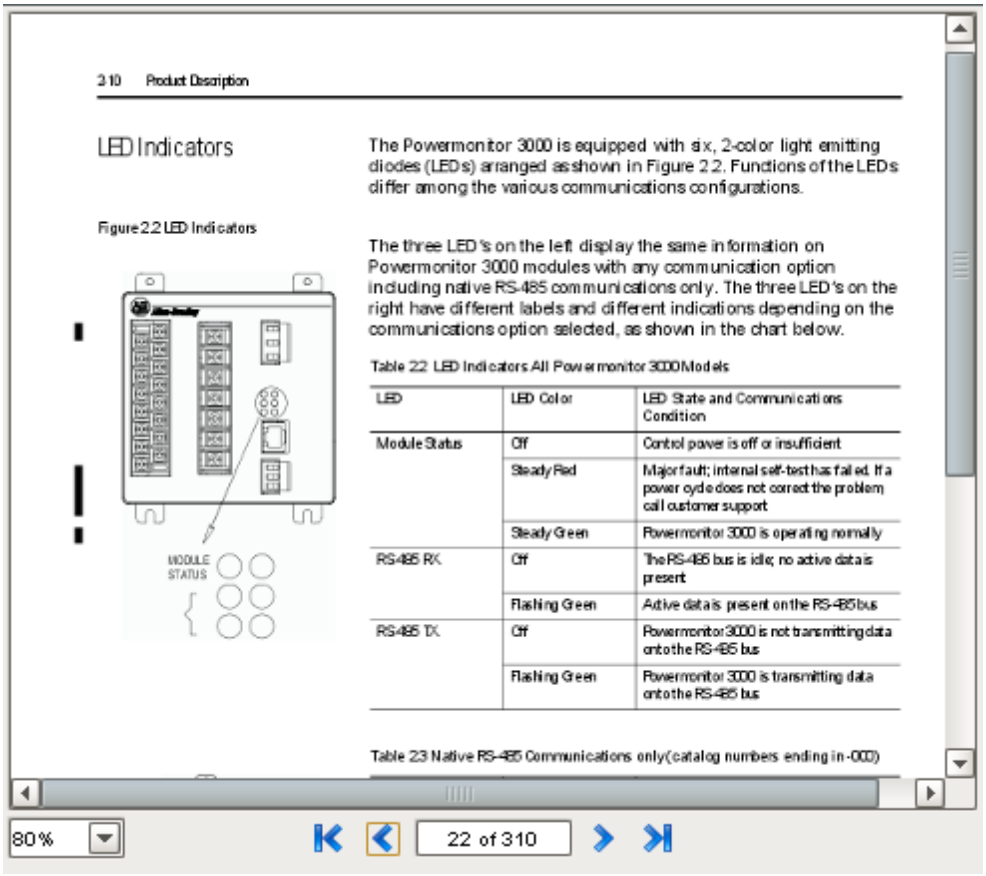
The following event sets are fired by this component. See Component Event Handlers to learn more.

mouse
mouseMotion
propertyChange

Scripting Functions

This component has no special scripting functions.

7.9.5 PDF Viewer


2 10 Product Description
LED Indicators
The Powermonitor 3000 is equipped with six, 2-color light emitting diodes (LEDs) arranged as shown in Figure 2.2. Functions of the LEDs differ among the various communications configurations.
Figure 2.2 LED Indicators
The three LED's on the left display the same information on Powermonitor 3000 modules with any communication option including native RS-485 communications only. The three LED's on the right have different labels and different indications depending on the communications option selected, as shown in the chart below.
Table 2.2 LED Indicators All Powermonitor 3000 Models
LED
LED Color
LED State and Communications Condition
Module Status
Off
Control power is off or insufficient
Steady Red
Major fault; internal self-test has failed. If a power cycle does not correct the problem call customer support
Steady Green
Powermonitor 3000 is operating normally
RS485 RX
Off
The RS-485 bus is idle; no active data is present
Flashing Green
Active data is present on the RS-485 bus
RS485 TX
Off
Powermonitor 3000 is not transmitting data onto the RS-485 bus
Flashing Green
Powermonitor 3000 is transmitting data onto the RS-485 bus
Table 2.3 Native RS-485 Communications only (catalog numbers ending in -000)
MODULE STATUS
80%
22 of 310

The PDF Viewer showing a schematic in a maintenance manual

Description

The PDF Viewer component displays a PDF that exists as a file in some accessible filesystem, or as a URL. Note that this component is simply for *viewing existing* PDFs. To *create* dynamic reports, use the Report Viewer component.

This component is typically used in conjunction with the File Explorer component, in order to create a PDF viewing window. See the File Explorer's documentation for instructions on how to put these two components together.

Warning. This component is *not* as high-quality as Adobe Reader. This component can only be guaranteed to correctly display reports *generated* by the Report Viewer. In practice, it is able to view many PDFs, but it does have trouble with some, especially PDFs created by AutoCAD. If this is a problem, use the free ActiveX module to embed an Adobe Reader control within your window. Of course, this will make your clients Windows-only.

Properties

Appearance

Font — Font of text of this component

Scripting name	`font`
Data type	Font

Foreground Color — The foreground color of the component.

Scripting name	`foreground`
Data type	Color

Background Color — The background color of the component.

Scripting name	`background`
Data type	Color

Zoom Factor — Sets the zoom factor for the report viewer

Scripting name	`zoomFactor`
Data type	float
Flags	bindable

Behavior

Print Mode — Sets the printing mode. Vector is fast and high-quality for printers that support it, but Raster mode can help the spool size with older printers.

Scripting name	`printingMode`
Data type	int
Flags	expert
Values	`0` Vector
	`1` Raster

Raster DPI — If the mode is raster, this is the DPI that will be used.

Scripting name	`printingDPI`
Data type	int
Flags	expert

Common

Name — The name of this component.

Scripting name	`name`
Data type	String
Flags	bindable

© 2011 Inductive Automation

Enabled — If disabled, a component cannot be used.
Scripting name: `enabled`
Data type: boolean

Visible — If disabled, the component will be hidden.
Scripting name: `visible`
Data type: boolean

Border — The border surrounding this component. NOTE that the border is unaffected by rotation.
Scripting name: `border`
Data type: Border

Mouseover Text — The text that is displayed in the tooltip which pops up on mouseover of this component.
Scripting name: `toolTipText`
Data type: String

Opaque — If false, backgrounds are not drawn. If true, backgrounds are drawn.
Scripting name: `opaque`
Data type: boolean

Data

Filename — The filename (or URL) of the PDF to view.
Scripting name: `filename`
Data type: String
Flags: bindable

Scripting

Events

The following event sets are fired by this component. See Component Event Handlers to learn more.

- mouse
- mouseMotion
- propertyChange

Scripting Functions

This component has no special scripting functions.

© 2011 Inductive Automation

Appendix B. Expression Functions

Part VIII

8 Appendix B. Expression Functions

8.1 Aggregates

8.1.1 groupConcat

groupConcat(dataset, column, separator**)**

Concatenates all of the values in the given `column` of the given `dataset` into a string, with each value separated by the string `separator`. Any null values in the column are ignored.

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
groupConcat({Root Container.Table.data}, 1, " / ")
```
... would return the string: `"380 / 120 / 125 / 322"`

```
groupConcat({Root Container.Table.data}, "ProductCode", ", ")
```
... would return the string: `"BAN_002, BAN_010, APL_000, FWL_220"`

8.1.2 max

max(dataset, column OR number, number...**)**

Finds and returns the maximum value in the given `column` of the given `dataset`, *or* the max value in a series of numbers specified as arguments. When looking up the max in a dataset, the column may be specified as an index or as a column name. Any `null` values in the column are ignored. If there are no rows in the dataset, zero is returned.

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
max({Root Container.Table.data}, 1)
```
... would return 380

You can also use this function to find the maximum in fixed series of numbers, specified as arguments, like this:

```
max(0, 10/2, 3.14)
```
... would return 5.

The following example is a great way to make sure a value never goes below zero:

```
max({SomeValue}, 0}
```

© 2011 Inductive Automation

8.1.3 maxDate

maxDate(dataset, columnIndex OR date, date...**)**

Finds and returns the maximum date in the given `column` of the given `dataset`, *or* the max value in a series of dates specified as arguments. When looking up the max date in a dataset, the column may be specified as an index or as a column name. Any `null` values in the column are ignored. If there are no rows in the dataset, `null` is returned.

For example, suppose you had a Table with this dataset in it:

AlertTime	Path	Severity
2010-01-08 7:28:04	Tanks/Tank5/TempHiAlert	4
2010-01-08 10:13:22	Tanks/Tank38/LoLevel	2
2010-01-08 13:02:56	Valves/Valve2/	2

You could use this expression to get the date and time for the most recent alert:

```
maxDate({Root Container.Table.data}, "AlertTime")
```

8.1.4 mean

mean(dataset, column OR number, number...**)**

Calculates the mean (a.k.a average) for the numbers in the given `column` of the given `dataset` *or* the mean of a series of numbers specified as arguments. When looking up the mean in a dataset, the column may be specified as an index or as a column name. Any `null` values in the column are ignored. If there are no rows in the dataset, zero is returned.

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
mean({Root Container.Table.data}, "Weight")
```

... would return 5.58675

```
mean(1,2,3)
```

... would return 2

8.1.5 median

median(dataset, column OR number, number...**)**

Calculates the median for the numbers in the given `column` of the given `dataset` *or* the median of a series of numbers specified as arguments. When looking up the median in a dataset, the column may be specified as an index or as a column name. Any `null` values in the column are ignored. If there are no rows in the dataset, zero is returned.

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287

© 2011 Inductive Automation

FWL_220	322	7.889

```
median({Root Container.Table.data}, "Weight")
```
... would return 5.566

```
median(1,2,3,3,10)
```
... would return 3

8.1.6 min

min(dataset, column OR number, number...**)**

Finds and returns the minimum value in the given `column` of the given `dataset`, *or* the min value in a series of numbers specified as arguments. When looking up the min in a dataset, the column may be specified as an index or as a column name. Any `null` values in the column are ignored. If there are no rows in the dataset, zero is returned.

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
min({Root Container.Table.data}, 1)
```
... would return 120

You can also use this function to find the minimum in fixed series of numbers, specified as arguments, like this:

```
min(0, 10/2, 3.14)
```
... would return 0.

The following example is a great way to make sure a value never goes above 180:

```
min({SomeValue}, 180}
```

8.1.7 minDate

minDate(dataset, columnIndex OR date, date...**)**

Finds and returns the minimum date in the given `column` of the given `dataset`, *or* the min value in a series of dates specified as arguments. When looking up the min date in a dataset, the column may be specified as an index or as a column name. Any `null` values in the column are ignored. If there are no rows in the dataset, `null` is returned.

For example, suppose you had a Table with this dataset in it:

AlertTime	Path	Severity
2010-01-08 7:28:04	Tanks/Tank5/TempHiAlert	4
2010-01-08 10:13:22	Tanks/Tank38/LoLevel	2
2010-01-08 13:02:56	Valves/Valve2/	2

You could use this expression to get the date and time for the oldest alert:

```
minDate({Root Container.Table.data}, "AlertTime")
```

© 2011 Inductive Automation

8.1.8 stdDev

stdDev(dataset, column OR number, number...)

Calculates the standard deviation of the values in the given `column` of the given `dataset`, *or* the standard deviation for a series of numbers specified as arguments. When looking up the standard deviation in a dataset, the column may be specified as an index or as a column name. Any `null` values in the column are ignored. If there are no rows in the dataset, zero is returned.

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
stdDev({Root Container.Table.data}, "Weight")
```

... would return 4.00532

8.1.9 sum

sum(dataset, column OR number, number...)

Calculates the sum of the values in the given `column` of the given `dataset`, *or* the sum for a series of numbers specified as arguments. When looking up the sum in a dataset, the column may be specified as an index or as a column name. Any `null` values in the column are ignored. If there are no rows in the dataset, zero is returned.

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
sum({Root Container.Table.data}, 1)
```

... would return 947

```
sum(1,2,3)
```

... would return 6

8.2 Colors

8.2.1 brighter

brighter(color)

Returns a color that is one shade brighter than the color given as an argument. Note that if you pass in a fully saturated color, like (255,0,0), it cannot be made brighter.

```
brighter(color(100,150,250))
```

... returns the color (142,214,255)

© 2011 Inductive Automation

8.2.2 color

color(red, green, blue, [alpha]**)**

Creates a color using the given red, green, and blue amounts, which are integers between 0-255. The optional alpha channel to the color controls transparency.

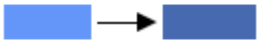
See also:
toColor

8.2.3 darker

darker(color**)**

Returns a color that is one shade darker than the color given as an argument.

```
brighter(color(100,150,250))
```
... returns the color (70,105,175)



8.2.4 gradient

gradient(number, low, high, lowColor, highColor**)**

Calculates a percentage given the three numeric arguments `number`, `low`, and `high`. Uses this percentage to create a color that is a mix between the two colors.

```
gradient(0, 0, 100, toColor("red"), toColor("blue"))
```
...returns red.

```
gradient(100, 0, 100, toColor("red"), toColor("blue"))
```
...returns blue.

```
gradient(60, 0, 100, toColor("red"), toColor("blue"))
```
...returns a shade of purple.

```
gradient({Root Container.Tank.value}, 0, 100, color(255,0,0), color(0,0,255))
```
...will return a gradient from red to blue based on the level of a tank.

8.3 Date and Time

8.3.1 dateArithmetic

dateArithmetic(date, number, field**)**

Adds or subtracts some amount of time from a date, returning the resulting date. The field argument must be a string, and must be one of these options:

"second"	"hr"
"sec"	"day"
"minute"	"week"
"min"	"month"
"hour"	"year"

```
dateArithmetic(toDate("2010-01-04 8:00:00"), 5, "hour")
```

© 2011 Inductive Automation

...returns the date '2010-01-04 13:00:00'

```
dateArithmetic({Root Container.DatePicker.date}, -8, "days")
```
...returns a date eight days before the date in a Popup Calendar component.

8.3.2 dateDiff

dateDiff(date, date, field**)**

Calculates the difference between the two dates, returning the result as a floating point value in the units specified by `field`, which must be a string matching one of these values:

"second"	"hr"
"sec"	"day"
"minute"	"week"
"min"	"month"
"hour"	"year"

The return value will be a floating point value, meaning that parts of units are considered. The exception to this rule is for the months and years fields, which will always return an integral difference. If the second `date` argument is after the first, the return value will be positive, otherwise it will be negative.

```
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-2-24 8:15:30"), "minute")
```
...returns 15.5

```
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-3-12 9:28:00"), "month")
```
...returns 1.0

```
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-3-12 9:28:00"), "day")
```
...returns 17.02

8.3.3 dateExtract

dateExtract(date, field**)**

Returns an integer value that is the value of the specified date field within the given date. The field must be a string, and must match one of these values:

"second"	"hr"
"sec"	"day"
"minute"	"week"
"min"	"month"
"hour"	"year"

Note: months are returned *zero-indexed*. That is, January is month 0, February is month 1, and so on. If this is inconvenient for you - just add one to the results.

```
dateExtract(toDate("2003-9-14 8:00:00"), "year")
```
...returns 2003

```
dateExtract(toDate("2009-1-15 8:00:00"), "month")
```
...returns 0

```
dateExtract(toDate("2008-1-24 8:00:00"), "month") + 1
```
...returns 1

© 2011 Inductive Automation

8.3.4 dateFormat

dateFormat(date, pattern**)**

Returns the given date as a string, formatted according to a pattern. The pattern is a format that is full of various placeholders that will display different parts of the date. These are case-sensitive! The most common placeholders are:

- **y** Year
- **M** Month
- **d** Day
- **E** Day of Week
- **a** am/pm marker
- **H** Hour of day (0-23)
- **h** Hour in am/pm (1-12)
- **m** Minute
- **s** Second
- **S** Millisecond
- **z** Time zone

These placeholders can be repeated for a different effect. For example, M will give you 1-12, MM will give you 01-12, MMM will give you Jan-Dec, MMMM will give you January-December.

```
dateFormat(toDate("2003-9-14 8:00:00"), "yyyy-MM-dd h a")
```
...returns the string "2003-09-14 8 am"

```
dateFormat(toDate("2003-9-14 8:00:00"), "MMM d, yyyy")
```
...returns the string "Sep 14, 2003"

8.3.5 now

now([pollRate]**)**

Returns the current time. The host computer's system clock is used, meaning that if this expression is being evaluated in a running client, the computer running the client's system clock is used. Note that this function is one of the few expression functions that will *poll*. If you do not specify a `pollRate`, it will default to 1,000ms. If you do not want this function to poll, use a poll rate of zero.

```
now()
```
...returns the current time, updates every second.

```
dateFormat(now(), "MMM d, h:mm a")
```
...returns a string representing the current time, formatted like "Feb 12, 9:54 AM"

8.3.6 timeBetween

timeBetween(date,date,date**)**

Checks to see if the given time is between the start and end times. The given times are expected as strings, and may include dates. Note: dates will be parsed according to the default system culture.

```
timeBetween(toDate("2003-9-14 12:00:00"), toDate("2003-9-14 8:00:00"), toDate("2003-9-14 18:00:
```
...returns true

```
timeBetween("2:00:00 pm", "9:00:00 am", "5:00:00 pm")
```
...returns true

© 2011 Inductive Automation

```
timeBetween("6/10/2006 2:00:00", "06/3/06 9:00:00", "2006-06-12 5:00:00")
```
...returns true (Note: This example also shows the variety of date formats accepted)

```
timeBetween(toDate("2003-9-14 20:00:00"), toDate("2003-9-14 18:00:00"), toDate("2003-9-15 2:00:
```
...returns true

8.4 Logic

8.4.1 binEnc

binEnc(boolean1, boolean2, ...**)**

This function, whose name stands for "binary encoder", takes a list of booleans, and treats them like the bits in a binary number. It returns an integer representing the decimal value of the number. The digits go from least significant to most significant.

This can be a very handy tool to convert bits into an integer code to drive the Component Styles feature.

```
binEnc(0,0,1,0)
```
...returns 4 (the value of 0100)

```
binEnc(true,0,1,1,0)
```
...returns 13 (the value of 01101)

8.4.2 binEnum

binEnum(boolean1, boolean2, ...**)**

This function, whose name stands for "binary enumeration", takes a list of booleans, and returns the index (starting at 1) of the first parameter that evaluates to true.

This can be a very handy tool to convert bits into an integer code to drive the Component Styles feature.

```
binEnum(0, 1, 0)
```
...returns 2

```
binEnum(0, false, 15, 0, 23)
```
...returns 3 (the index of the 15 - any non-zero number is "true")

8.4.3 coalesce

coalesce(value1, value2, ...**)**

This function, which accepts any number of arguments, evaluates each in order, and returns the first non-null argument. Typically, you would call this with two arguments - the first being something dynamic, the second being a static value to use as a guard in case the dynamic value is null. The function itself detects its return type based on the type of the last argument.

```
coalesce(null, "abc")
```
...would return "abc"

```
coalese("xyz", "abc")
```

© 2011 Inductive Automation

...would return "xyz"

```
coalese({Root Container.MyDataSet}["ColumnName"], 0)
```
...would return the value in the dataset if it isn't null, but 0 if it is null.

8.4.4 getBit

getBit(number, position)

This function returns the bit value (an integer, 0 or 1) in the `number` at position `position`, according to its binary representation. The least significant bit in a number is position 0.

```
getBit(0,0)
```
...would return 0

```
getBit(1,0)
```
...would return 1

```
getBit(8,3)
```
...would return 1

```
getBit(8,2)
```
...would return 0

8.4.5 if

if(condition, trueReturn, falseReturn)

This function evaluates the expression `condition`, and returns the value of `trueReturn` or `falseReturn` depending on the boolean value of `condition`.

```
if(1, "Yes", "No")
```
...would return "Yes"

```
if(0, "Yes", "No")
```
...would return "No"

```
if({Root Container.CheckBox.selected}, "Selected", "Not Selected")
```
...would return the a description of the state of the checkbox

8.4.6 isNull

isNull(value)

Tests to see whether or not the argument `value` is null or not. Note that you can also check for null by simply comparing the value to the null keyword. `isNull(x)` is the same as `x = null`.

```
if(isNull({Root Container.MyProperty}), 0, 1)
```
... returns 0 if the property is null, 1 otherwise. See also: coalesce.

8.4.7 lookup

lookup(dataset, lookupValue, noMatchValue, [lookupColumn], [resultColumn])

This looks for `lookupValue` in the `lookupColumn` of `dataset`. If it finds a match, it will return the value from the `resultColumn` on the same row as the match. If no match is found,

© 2011 Inductive Automation

noMatchValue is returned. Note: The type of the value returned will always be coerced to be the same type as the noMatchValue.

If lookupColumn is not specified, it defaults to 0. If resultColumn is not specified, it defaults to 1.

The examples are based of a table that has the following data in it:

PRODUCT	PRICE	CATEGORY
"Apples"	1.99	"Fruit"
"Carrots"	3.50	"Vegetable"
"Walnuts"	6.25	"Nut"

```
lookup({Root Container.Table.data}, "Carrots", -1.0)
```
... returns 3.50

```
lookup({Root Container.Table.data}, "Grapefruit", -1)
```
... returns -1, the noMatchValue

```
lookup({Root Container.Table.data}, "Walnuts", "Unknown", 0, "Category")
```
... returns "Nut"

```
lookup({Root Container.Table.data}, "Pecans", "Unknown", 0, 2)
```
... returns "Unknown", the noMatchValue

8.4.8 switch

switch(value, case1, ...caseN, return1, ...returnN, returnDefault**)**

This function acts like the switch statement in C-like programming languages. It takes the value argument and compares it to each of the case1 through caseN expressions. If value is equal to caseX, then switch returns valueX. If value is not equal to any of the case1..N, then returnDefault is returned.

```
switch(
15, // value
1,  // case 1
24, // case 2
15, // case 3
44, // return 1
45, // return 2
46, // return 3
-1) // default
```
...would return 46 because the value (15) matched case 3, so the third return (46) was returned.

```
switch(
35, // value
50,  // case 1
52, // case 2
200, // return 1
100, // return 2
-1) // default
```
...would return -1 because the value (35) didn't match case 1 or 2, so the returnDefault was used.

```
switch(
```

© 2011 Inductive Automation

```
1, // value
0, 1, 2,  // cases 1-3
"Off", // return 1
"Running", // return 2
"Fault", // return 3
forceQuality("!BAD STATE!",0)) // default
```

...would return "Running".

8.4.9 try

try(expression, failover**)**

This expression is used to swallow errors caused by other expressions. The first expression will be executed, and if it executes successfully, its value will be used. However, if there is an error evaluating it, the value of failover will be used, with a data quality of 310 (EXPRESSION_EVAL_ERROR).

```
try(toInteger("boom"), -1)
```

... returns -1 with a quality code of 310

8.5 Math

8.5.1 abs

abs(number**)**

Returns the absolute value of `number`.

```
abs(-4)
```

... returns 4

8.5.2 acos

acos(number**)**

Returns the arc cosine of `number`, which must be a number between -1 and 1. The results will be an angle expressed in radians in the range of `0.0` through *pi*.

```
acos(.38)
```

... returns 1.181

8.5.3 asin

asin(number**)**

Returns the arc sine of `number`, which must be a number between -1 and 1. The results will be an angle expressed in radians in the range of `-`*pi*`/2` through *pi*`/2`

```
asin(.38)
```

... returns 0.3898

8.5.4 atan

atan(number**)**

Returns the arc tangent of `number`, which must be a number. The results will be an angle expressed

in radians in the range of `-pi/2` through `pi/2`

```
atan(.38)
```
... returns 0.3631

8.5.5 ceil

```
ceil(number)
```

Returns the smallest floating point value that is greater than or equal to the argument and is equal to a mathematical integer.

```
ceil(2.38)
```
... returns 3.0

8.5.6 cos

```
cos(number)
```

Returns the trigonometric cosine of `number`, which is interpreted as an angle expressed in radians. The results will be a floating point value.

```
cos(1.89)
```
... returns -0.31381

8.5.7 exp

```
exp(number)
```

Returns Euler's number *e* raised to the power of the argument `number`, or e^{number}

```
exp(5)
```
... returns 148.4

8.5.8 floor

```
floor(number)
```

Returns the largest floating point value that is less than or equal to the argument and is equal to a mathematical integer.

```
floor(2.72)
```
... returns 2.0

8.5.9 log

```
log(number)
```

Returns the natural logarithm (base *e*) of a `number`.

```
log(28)
```
... returns 3.332

© 2011 Inductive Automation

8.5.10 round

round(number, [decimals]**)**

Rounds a floating point number. If the decimals argument is omitted, then the number is rounded to the nearest integer value, and the result will be a long (64-bit integer).

If a number of decimal places are specified, the result will be a double (64-bit floating point value), and the result will be rounded to the given number of decimal places.

```
round(3.829839, 2)
```
... returns 3.83

8.5.11 sin

sin(number**)**

Returns the trigonometric sine of `number`, which is interpreted as an angle expressed in radians. The results will be a floating point value.

```
sin(1.89)
```
... returns 0.9495

8.5.12 sqrt

sqrt(number**)**

Returns the square root of the argument `number`.

```
sqrt(64)
```
... returns 8.0

8.5.13 tan

tan(number**)**

Returns the trigonometric tangent of `number`, which is interpreted as an angle expressed in radians. The results will be a floating point value.

```
tan(1.89)
```
... returns -3.026

8.5.14 todegrees

todegrees(number**)**

Converts an angle measured in radians to an equivalent angle measured in degrees.

```
toDegrees(3.14)
```
... returns 179.9088

8.5.15 toradians

toradians(number**)**

Converts an angle measured in degrees to an equivalent angle measured in radians.

© 2011 Inductive Automation

```
toRadians(180)
```
... returns 3.141592653589793

8.6 Strings

8.6.1 concat

concat(string1, string2, ...**)**

Concatenates all of the strings passed in as arguments together. Rarely used, as the + operator does the same thing.

```
concat("The answer is: ", "42")
```
... returns "The answer is 42"

8.6.2 escapeSQL

escapeSQL(string**)**

Returns the given string with special SQL characters escaped. This is a fairly simplistic function - it just replaces single quotes with two single quotes, and backslashes with two backslashes. See system.db.runPrepUpdate for a much safer way to sanitize user input.

```
"SELECT * FROM mytable WHERE option = '" + escapeSQL("Jim's Settings") + "'"
```
... returns SELECT * FROM mytable WHERE option='Jim''s Settings'

```
"SELECT * FROM mytable WHERE option = '" + escapeSQL({Root Container.TextField.text}) + "'"
```
... returns a query with sanitized user input from a text field.

8.6.3 escapeXML

escapeXML(string**)**

Returns the given string after being escaped to be valid for inclusion in XML. This means replacing XML special characters with their XML entity equivalents.

```
escapeXML("Use Navigate > PB to get to the Pork&Beans section.")
```
... returns "Use Navigate > PB to get to the Pork&Beans section."

8.6.4 indexOf

indexOf(string, substring**)**

Searches for the first occurrence of the substring inside of string. Returns the index of where substring was found, or -1 if it wasn't found.

```
indexOf("Hamburger", "urge")
```
...returns 4

```
indexOf("Test", "")
```
...returns 0

```
indexOf("Disfunctional", "fun")
```
...returns 3

© 2011 Inductive Automation

```
indexOf("Disfunctional", "marble")
```
...returns -1

```
indexOf("banana", "n")
```
...returns 2

8.6.5 lastIndexOf

lastIndexOf(string, substring**)**

Searches for the last occurrence of the `substring` inside of `string`. Returns the index of where `substring` was found, or -1 if it wasn't found.

```
indexOf("Hamburger", "urge")
```
...returns 4

```
indexOf("Test", "")
```
...returns 4

```
indexOf("Disfunctional", "fun")
```
...returns 3

```
indexOf("Disfunctional", "marble")
```
...returns -1

```
indexOf("banana", "n")
```
...returns 4

8.6.6 left

left(string, charCount**)**

Returns `count` characters from the left side of `string`, where `count` and `string` are the arguments to the function.

```
left("hello", 2)
```
...returns `"he"`

```
left("hello", 0)
```
...returns `""`

```
left("hello", 5)
```
...returns `"hello"`

8.6.7 len

len(value**)**

Returns the length of the argument, which may be a string or a dataset. If the argument is a string, it returns the number of characters in the string. If the argument is a dataset, it returns the number of rows in the dataset. Will return zero if the argument is null.

```
len("Hello World")
```
... returns 11

© 2011 Inductive Automation

```
len({Root Container.Table.data})
```
... returns the number of rows in the table.

8.6.8 lower

lower(string)

Takes a string and returns a lower-case version of it.

```
lower("Hello World")
```
... returns "hello world"

8.6.9 numberFormat

numberFormat(number, pattern)

Returns a string version of the `number` argument, formatted as specified by the `pattern` string. This is commonly used to specify the number of decimal places to display, but can be used for more advanced formatting as well. The pattern string is a numeric format string, which may include any of these characters that instruct it how to format the number.

0	Specifies a required digit	**E**	Scientific notation
#	Specifies an optional digit	**;**	Used to separate positive and negative patterns
,	The grouping separator	**%**	Multiplies the value by 100 and shows as a percent
-	A minus sign	**'**	Used to quote special characters

This table shows some numbers, and the result of using various format strings to format them.

Number	Pattern	Result
5	0	5
5	0.0	5.0
5	00.0	05.0
123	#,##0	123
1024	#,##0	1,024
1337	#,##0.#	1,337
1337.42	#.##0.#	1,337.4
87.32	#,##0.0000	87.3200
-1234	#,##0	-1,234
-1234	#,##0;(#)	(1,234)
4096	0.###E0	4.096E3
.348	#.00%	34.80%
34.8	#0.00'%'	34.80%

Example:
```
numberFormat(34.8, "#0.00'%'")
```
... returns the string "34.80%"

8.6.10 repeat

repeat(string, count)

Repeats the given string some number of times.

```
repeat("hello", 2)
```

© 2011 Inductive Automation

...returns "hellohello"

```
repeat("hello", 0)
```
...returns ""

8.6.11 replace

replace(string, string, string**)**

Finds all occurrences of a substring inside of a source string, and replaces them with the replacement string. The first argument is the source, the second is the search string, and the third is the replacement.

```
replace("XYZ", "Y", "and")
```
...returns "XandZ"

```
repeat("bob and mary went to bob's house", "bob", "judith")
```
...returns "judith and mary went to judith's house"

8.6.12 right

right(string, charCount**)**

Returns `count` characters from the right side of `string`, where `count` and `string` are the arguments to the function.

```
right("hello", 2)
```
...returns "lo"

```
right("filename.pdf", 3)
```
...returns "pdf"

```
right("hello", 0)
```
...returns ""

8.6.13 split

split(string, regex, [limit]**)**

This function takes the string `string` and splits it into a bunch of substrings. The substrings are return as a dataset with one column called "parts". The split occurs wherever the regular expression `regex` occurs. Don't be intimidated by the regular expression, this is normally just another string, like "," for comma separated lists.

The optional `limit` argument, if greater than zero, limits the number of times the regex pattern is applied to limit-1. Put another way, it limits the length of the resulting dataset to length limit. If limit is non-positive then the regex pattern will be applied as many times as possible and the returned dataset can have any length. If limit is zero (the default) then the pattern will be applied as many times as possible, the returned dataset can have any length, and trailing empty strings will be discarded.

```
split("hello,world", ",")
```
... returns

parts

© 2011 Inductive Automation

"hello"
"world"

```
split("boo:and:foo", ":")
```
... returns

parts
"boo"
"and"
"foo"

```
split("boo:and:foo", ":", 2)
```
... returns

parts
"boo"
"and:foo"

8.6.14 stringFormat

```
stringFormat(format, args...)
```

--MISSING--

8.6.15 substring

```
substring(string, startIndex, [endIndex])
```

Substring will return the portion of the string from the `startIndex` to the `endIndex`, or end of the string if `endIndex` is not specified. All indexes start at 0, so in the string "Test", "s" is at index 2.

```
substring("unhappy", 2)
```
... returns `"happy"`

```
substring("hamburger", 4, 8)
```
... returns `"urge"`

8.6.16 trim

```
trim(string)
```

Takes the argument string and trims of any leading and/or trailing whitespace, returning the result.

```
trim("Hello Dave  ")
```
... returns `"Hello Dave"`

```
trim(" Goodbye.")
```
... returns `"Goodbye."`

8.6.17 upper

```
upper(string)
```

Takes a string and returns an upper-case version of it.

© 2011 Inductive Automation

```
upper("Hello World")
```
... returns "HELLO WORLD"

8.7 Type Casting

8.7.1 toBoolean

toBoolean(value, [failover]**)**

Tries to convert `value` to a boolean, according to these rules:
1. If `value` is a number, 0 is `false` and anything else is `true`.
2. If `value` is a string, then the strings (case insensitive) "on", "true", "t", "yes", "y" are all `true`. The strings (case insensitive) "off", "false", "f", "no", "n" are considered `false`. If the string represents a number, the first rule applies. All other strings fail type casting.
3. All other types fail type casting.

If type casting fails, an error is thrown, unless the `failover` argument is specified, in which case it will be used.

```
toBoolean(1)
```
... returns `true`.

```
toBoolean("abc", false)
```
... returns `false`

8.7.2 toBorder

toBorder(value, [failover]**)**

Takes a string and tries to convert it into a border. The string must be a semi-colon separated list of values. The first value is the name of the border. The other values depend on the type of border. The following table defines the border types and the arguments they accept.

Border Type	Options
`bevel`	**bevelType** Bevel Types: 0 = Raised 1 = Lowered 1010 = Double
`button`	*none*
`etched`	**etchType** Etch Types: 0 = Raised 1 = Lowered
`etchedtitled`	**title**; **style**; **fontJustification**; **fontPosition**; **fontColor**; **font** Styles: 0 = Etched / Lowered 1 = Etched / Raised 2 = Beveled / Lowered 3 = Beveled / Raised 4 = Develed / Double 5 = Standard
`field`	*none*
`line`	**color**; **thickness**

© 2011 Inductive Automation

`linetitled`	**title**; **width**; **lineColor**; **fontJustification**; **fontPosition**; **fontColor**; **font**
`matte`	**color**; **topWidth**, **leftWidth**; **bottomWidth**; **rightWidth**
`paneltitled`	**title**; **style**; **mainColor**; bgColor, shadowSize, **fontJustification**; **fontPosition**; **fontColor**; **font** Styles: 0=Gradient / South-to-North 1=Gradient / West-to-East 2=Gradient / North-to-South 3=Gradient / East-to-West 4=Solid

Other Constants			
Font Justifications:	1 = Left 2 = Center 3 = Right 4 = Leading 5 = Trailing	Font Positions:	1 = Above Top 2 = Top 3 = Below Top 4 = Above Bottom 5 = Bottom 6 = Below Bottom

Examples:

```
toBorder("bevel;1010")
```

... returns

```
toBorder("matte;red;10;1;1;1")
```

... returns

```
toBorder("paneltitled;MyTitle")
```

... returns


```
toBorder("paneltitled;Options;1;lightgray;gray;0;;;(0,255,0)")
```

... returns

8.7.3 toColor

`toColor(`value, [failover]**`)`**

This function tries to convert value to a color. It assumes that `value` is a string. If you have integers representing Red, Green, and Blue values see the color expression. The string `value` is converted to a color according to these rules:

1. If value is a name of a color as defined in the table below, the corresponding color will be returned. Note that color names are case insensitive.
2. If value is a hex color string (with or without a leading "#", the color equivalent of that hex string will be used. Examples: "#FF0000", "556B2F"

© 2011 Inductive Automation

3. If value is a list of 3 or 4 integers, a color will be created that uses the first three integers as red, green, and blue values, and the optional fourth integer as an alpha channel value. All values should be between 0 and 255. The list is free-form, any non-digit characters may be used as delimiters between the digits. Examples: "(0,0,0)", "23-99-203", "[255,255,33,127]"

For example, all of these expressions return the color red:

```
toColor("red")
toColor("#FF0000")
toColor("255,0,0")
```

You can use the `failover` parameter to ensure that this expression returns something even if the input string may be bad:

```
toColor({UserOptions/CustomColor}, "black")
```

Named Colors		
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFD4	
Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Clear	(transparent)	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
Darkorange	#FF8C00	
DarkOrchid	#9932CC	
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBC8F	

© 2011 Inductive Automation

DarkSlateBlue	#483D8B
DarkSlateGray	#2F4F4F
DarkTurquoise	#00CED1
DarkViolet	#9400D3
DeepPink	#FF1493
DeepSkyBlue	#00BFFF
DimGray	#696969
DodgerBlue	#1E90FF
Feldspar	#D19275
FireBrick	#B22222
FloralWhite	#FFFAF0
ForestGreen	#228B22
Fuchsia	#FF00FF
Gainsboro	#DCDCDC
GhostWhite	#F8F8FF
Gold	#FFD700
GoldenRod	#DAA520
Gray	#808080
Green	#008000
GreenYellow	#ADFF2F
HoneyDew	#F0FFF0
HotPink	#FF69B4
IndianRed	#CD5C5C
Indigo	#4B0082
Ivory	#FFFFF0
Khaki	#F0E68C
Lavender	#E6E6FA
LavenderBlush	#FFF0F5
LawnGreen	#7CFC00
LemonChiffon	#FFFACD
LightBlue	#ADD8E6
LightCoral	#F08080
LightCyan	#E0FFFF
LightGoldenRodYellow	#FAFAD2
LightGreen	#90EE90
LightGrey	#D3D3D3
LightPink	#FFB6C1
LightSalmon	#FFA07A
LightSeaGreen	#20B2AA
LightSkyBlue	#87CEFA
LightSlateBlue	#8470FF
LightSlateGray	#778899
LightSteelBlue	#B0C4DE
LightYellow	#FFFFE0
Lime	#00FF00
LimeGreen	#32CD32
Linen	#FAF0E6
Magenta	#FF00FF
Maroon	#800000
MediumAquaMarine	#66CDAA

© 2011 Inductive Automation

MediumBlue	#0000CD
MediumOrchid	#BA55D3
MediumPurple	#9370D8
MediumSeaGreen	#3CB371
MediumSlateBlue	#7B68EE
MediumSpringGreen	#00FA9A
MediumTurquoise	#48D1CC
MediumVioletRed	#C71585
MidnightBlue	#191970
MintCream	#F5FFFA
MistyRose	#FFE4E1
Moccasin	#FFE4B5
NavajoWhite	#FFDEAD
Navy	#000080
OldLace	#FDF5E6
Olive	#808000
OliveDrab	#6B8E23
Orange	#FFA500
OrangeRed	#FF4500
Orchid	#DA70D6
PaleGoldenRod	#EEE8AA
PaleGreen	#98FB98
PaleTurquoise	#AFEEEE
PaleVioletRed	#D87093
PapayaWhip	#FFEFD5
PeachPuff	#FFDAB9
Peru	#CD853F
Pink	#FFC0CB
Plum	#DDA0DD
PowderBlue	#B0E0E6
Purple	#800080
Red	#FF0000
RosyBrown	#BC8F8F
RoyalBlue	#4169E1
SaddleBrown	#8B4513
Salmon	#FA8072
SandyBrown	#F4A460
SeaGreen	#2E8B57
SeaShell	#FFF5EE
Sienna	#A0522D
Silver	#C0C0C0
SkyBlue	#87CEEB
SlateBlue	#6A5ACD
SlateGray	#708090
Snow	#FFFAFA
SpringGreen	#00FF7F
SteelBlue	#4682B4
Tan	#D2B48C
Teal	#008080
Thistle	#D8BFD8

© 2011 Inductive Automation

Tomato	#FF6347	
Transparent	#FFFFFF	
Turquoise	#40E0D0	
Violet	#EE82EE	
VioletRed	#D02090	
Wheat	#F5DEB3	
White	#FFFFFF	
WhiteSmoke	#F5F5F5	
Yellow	#FFFF00	
YellowGreen	#9ACD32	

8.7.4 toDataSet

toDataSet(value, [failover]**)**

Tries to coerce `value` into a dataset. Not many things can be coerced into datasets. Namely, only DataSets and PyDataSets can be coerced into DataSets. This is useful for the runScript() expression, to convince the expression compiler to let you assign the return value of a scripting function to a DataSet property.

```
toDataSet(runScript("app.funcs.runSomeFunction()"))
```
... coerces the value returned by the a project scripting function into a dataset.

See also:
DataSets vs PyDataSets

8.7.5 toDate

toDate(value, [failover]**)**

Tries to coerce `value` into a Date. If `value` is a number or a string that represents a number, the number is treated as the number of milliseconds since the epoch, January 1, 1970, 00:00:00 GMT. If value is a string, it is parsed to see if it represents a date in one of these two formats: "yyyyMMdd.HHmmssSSSZ" or "yyyy-MM-dd HH:mm:ss". If not, type casting fails.
The failover value must be a number or string with the same restrictions.

```
toDate("2007-04-12 16:28:22")
```
... returns April 12th, 2007, 4:28:22 PM

8.7.6 toDouble

toDouble(value, [failover]**)**

Tries to coerce `value` into a double (64-bit floating point value). If `value` is a number, the conversion is direct. If `value` is a string, it is parsed to see if it represents a double. If not, type casting fails.

```
toDouble("38.772")
```
... returns 38.772

```
toDouble({Root Container.TextField.text}, 0.0)
```
... returns the value in the text box as a double, or 0.0 if the value doesn't represent an number.

© 2011 Inductive Automation

8.7.7 toFloat

toFloat(value, [failover]**)**

Tries to coerce `value` into a float (32-bit floating point vaule). If `value` is a number, the conversion is direct. If `value` is a string, it is parsed to see if it represents a float. If not, type casting fails.

```
toFloat("38.772")
```
... returns 38.772

```
toFloat({Root Container.TextField.text}, 0.0)
```
... returns the value in the text box as a float, or 0.0 if the value doesn't represent an number.

8.7.8 toFont

toFont(value, [failover]**)**

Coerces a string into a font. The string must be in the format:

`font(`*`fontName`*`, `*`fontType`*`, `*`fontSize`*`)`

fontName is the name of the font to use. Note that special care must be taken with fonts, because of the web-launched nature of the clients. You can only use font names that exist on the client machines. The following font names are known as *logical* fonts, meaning that they are guaranteed to exist on all systems, mapped to the most appropriate real, or *physical* font that exists on the host system:

- Serif
- SansSerif
- Monospaced
- Dialog
- DialogInput

fontType is a string, that should match one of these (case-insensitive):

- Plain
- Bold
- Italic
- BoldItalic

fontSize is an integer that represent the font's point size.

```
toFont("font(Dialog,Bold,12)")
```
... returns the standard font used in most clients.

8.7.9 toInt

toInt(value, [failover]**)**

Tries to coerce value into an integer (32-bit integer). If value is a number, the conversion is direct (with possible loss of precision). If value is a string, it is parsed to see if it represents an integer. If not, type casting fails. Will round if appropriate.

```
toInt("38")
```
... returns 38

```
toInt("33.9")
```

© 2011 Inductive Automation

... returns 34

```
toInt({Root Container.TextField.text}, -1)
```
... returns the value in the text box as an int, or -1 if the value doesn't represent an number.

8.7.10 toInteger

toInteger(value, [failover]**)**

Identical to the `toInt` expression function.

8.7.11 toLong

toLong(value, [failover]**)**

Tries to coerce value into a long (64-bit integer). If value is a number, the conversion is direct. If value is a string, it is parsed to see if it represents a long. If not, type casting fails. Will round if appropriate.

```
toLong("38")
```
... returns 38

```
toLong("33.9")
```
... returns 34

```
toLong({Root Container.TextField.text}, -1)
```
... returns the value in the text box as an long, or -1 if the value doesn't represent an number.

8.7.12 toStr

toStr(value, [failover]**)**

Identical to the `toString` expression function.

8.7.13 toString

toString(value, [failover]**)**

Represents the `value` as a string. Will succeed for any type of value.

```
toString(1/3.0)
```
... returns `"0.3333333333333333"`

```
toString({Root Container.Table.data})
```
... returns something like: `"Dataset [150R x 3C]"`

8.8 Advanced

8.8.1 forceQuality

forceQuality(value, [qualityCode]**)**

Returns the given value, but overwrites the *quality* of that value. If the quality argument is omitted, the quality will be GOOD (192). This is a way to have expressions opt-out of the quality overlay system. You can also force a specific quality code here by including the quality argument.

```
forceQuality({Tanks/Tank15})
```

© 2011 Inductive Automation

... returns the value of the Tank15 tag, but always with a good quality code.

```
forceQuality({Tanks/Tank15}, 410)
```
... returns the value of the Tank15 tag, but always with a TAG_DISABLED quality.

See also:
Quality Overlays

8.8.2 runScript

runScript(scriptFunction, [pollRate]**)**

Runs a single line of Python code as an expression. If a poll rate is specified, the function will be run repeatedly at the poll rate. This is a very powerful way for you to add extensions to the expression language.

For example, one could write a project script module function called `app.weather.getTempAt(zip)` that queried a web service for the current temperature at a given zipcode, and then bind the value of a label to the return value of that function.

You could implement `app.weather.getTempAt(zip)` with this *Python script*:

```
# This function would query Yahoo Weather for the temperature at
# the given zipcode and find the temperature using a regular expression
def getTempAt(zipCode):
    import system
    import re #Regular Expression library

    response = system.net.httpGet("http://xml.weather.yahoo.com/forecastrss?p=" + str(zipCode))

    # NOTE - if you've never seen regular expressions before, don't worry, they look
    # confusing even to people who use them frequently.
    pattern = re.compile('.*?<yweather:condition (.*?)/>', re.DOTALL)
    match = pattern.match(response)
    if match:
        subText = match.group(1)
        temp = re.compile('.*?temp="(.*?)"').match(subText).group(1)
        return int(temp)
    else:
        system.gui.errorBox("Yahoo weather service changed")
        return -1
```

And then you could use this *expression* to bind a property value to the weather:

```
runScript("app.weather.getTempAt('95818')", 15000)
```
... This would bind a property to the temperature in sunny Sacramento, CA, and would refresh itself every 15 seconds.

See also:
About Python

8.8.3 tag

tag(tagPath**)**

Returns the value of the tag at the path specified. Normally, you'd use the expression language's

built-in bound-value syntax to use a tag value in an expression. What makes this function useful is that the path itself can be the result of an expression, meaning it can be dynamic.

```
tag("Tanks/Tank5")
```
... returns Tank5's value.

```
tag("Tanks/Tank" + {Root Container.TankNum})
```
... returns the value for the tank represented by the dynamic property TankNum on the Root Container.

See also:
Indirect Tag Binding

© 2011 Inductive Automation

Appendix C. Scripting Functions

Part IX

9 Appendix C. Scripting Functions

9.1 About

The Ignition scripting API, which is available under the module name "system", is full of functions that are useful when designing projects in Ignition. From running database queries, manipulating components, to

Some of these functions only work in the Gateway scope, and other only work in the Client scope, while the rest will work in any scope.

"*I'm upgrading from FactoryPMI - will my calls to fpmi.* still work?*"
Yes. Ignition's scripting API is backwards compatible. You'll probably want to gradually move your "`fpmi`" references to "`system`" but you don't need to.

See also:
Gateway vs Client Scripts

9.2 system.alert

9.2.1 system.alert.acknowledgeAlert

Description

Acknowledges an alert, as specified by a `system`, `path`, and `stateName`. When run in a Client script, the currently logged-in user will be recorded as having acknowledged the alert. When run in a Gateway script, you must provide a username that will be recorded with the acknowledgement, since no user actually acknowledged the alert.

Syntax

```
system.alert.acknowledgeAlert(system, path, stateName)
```

Parameters
String `system` - The originating system for the alert being acknowledged.
String `path` - The path to the alert being acknowledged.
String `stateName` - The alert state name to acknowledge.

Returns
nothing

Scope
Client

```
system.alert.acknowledgeAlert(system, path, stateName, user)
```

Parameters
String `system` - The originating system for the alert being acknowledged.
String `path` - The path to the alert being acknowledged.
String `stateName` - The alert state name to acknowledge.
String `user` - A username to use for who acknowledged this alert. Only available in the Gateway-scoped version of this function.

Returns
nothing

© 2011 Inductive Automation

Scope
Gateway

Examples

This example shows the basic syntax for acknowledging an alert.

```
system.alert.acknowledgeAlert("SQLTags.default", "[default]Alm_ESTOP", "ALM_STOP")
```

This code snippet could be used as a `mouseReleased` event handler on a Table component whose `data` was the return value of the `system.alert.queryAlertStatus` function. It would present a right-click menu to acknowledge the currently selected alert.

```
row = event.source.selectedRow
if row != -1:
   data = event.source.data
   alertSys = data.getValueAt(row,"System")
   alertPath = data.getValueAt(row,"Path")
   alertState = data.getValueAt(row,"State Name")
   def ack(event, aSys=alertSys, aPath=alertPath, aState=alertState):
      import system
      system.alert.acknowledgeAlert(aSys,aPath,aState)

   menu = system.gui.createPopupMenu({"Acknowledge":ack})
   menu.show(event)
```

See also:
Event Types / Mouse Events
system.alert.queryAlertStatus
system.gui.createPopupMenu

9.2.2 system.alert.queryAlertHistory

Description

This function queries one of the configured Alert Storage profiles for alert history. The filter arguments help to narrow down the results to alerts that match various criteria, most commonly a range of dates. You can use `*` to match any number of characters and `?` to match a single character in the filter string arguments.

The results of this function are a dataset with the following columns:
System - The system that issued the alert.
Path - The path to the alert item
Display Path - The custom display path (if any) for the alert. Will be the Path if no Display Path is configured.
State Name - The state name for the alert.
Severity - The severity, as a string.
Severity Code - The severity as an integer. 0-4, low-high.
Active - A boolean indicating whether this alert state is still active.
Active Timestamp - The time at which this alert went active.
Active Value - The value that triggered this alert to go active.
Cleared - A boolean indicating whether this alert has cleared.
Cleared Timestamp - The time at which this alert cleared. May be null.
Cleared Value - The value that cleared the alert.
Acked - A boolean indicating whether or not this alert was been acknowledged.
Ack Timestamp - The time that the alert was acknowledged. May be null.

© 2011 Inductive Automation

Ack user - The user who acknowledged the alert.
Notes - The notes field for the alert
Flags - A bitmask representing the current alert state. 0x01= Active, 0x02=Cleared, 0x04=Acknowledged. So if the alert is active and acknowledged, but not cleared, this will be 0x01 | 0x04 = 5;

This function accepts *keyword-style* invocation. See also: Functions / Keyword Invocation

Syntax

```
system.alert.queryAlertHistory(storageProfile, startDate, endDate, system, path, stateName, minSeverity, maxSeverity, activeAndUnacked, activeAndAcked, clearAndUnacked, clearAndAcked, sortOrder, displayPath)
```

Parameters

String `storageProfile` - The name of the alert storage profile to query.
Date `startDate` - Earliest alert to return. Defaults to 8 hours earlier than current time if omitted.
Date `endDate` - Latest alert to return. Defaults to current time if omitted.
String `system` - Filter string to restrict results based on the alert system.
String `path` - Filter string to restrict results based on the alert path
String `stateName` - Filter string to restrict results based on the alert state name
Integer `minSeverity` - Minimum severity to return. Defaults to 0 (Low).
Integer `maxSeverity` - Maximum severity to return. Defaults to 4 (High).
Boolean `activeAndUnacked` - Whether or not to return alerts that are currently active and unacknowledged. Default is true.
Boolean `activeAndAcked` - Whether or not to return alerts that are currently active and have been acknowledged. Default is true.
Boolean `clearAndUnacked` - Whether or not to return alerts that are cleared and unacknowledged. Default is true.
Boolean `clearAndAcked` - Whether or not to return alerts that are cleared and have been acknowledged. Default is true.
String `sortOrder` - The sort order in which to return matching alerts. Either "asc" or "desc", referring to the alert's active timestamp. Default is "desc"
String `displayPath` - Filter string to restrict results based on the alert's display path

Returns

Dataset - A dataset containing the historical alert events from the given storage profile that matched the filter and date range arguments.

Scope

All

Examples

This code would query an alert storage profile called "DBHistory", looking for the number of unacknowledged alerts in the last 36 hours, displaying the number to the user in a popup message.

```
from java.util import Date
from java.util import Calendar

cal = Calendar.getInstance()

end = cal.getTime()
cal.add(Calendar.HOUR, -36)
start = cal.getTime()
```

© 2011 Inductive Automation

```
results = system.alert.queryAlertHistory("DBHistory", start, end, activeAndAcked=0, clearAndAck

if results.rowCount > 0:
   system.gui.messageBox("There are %d un-acked alerts!" % results.rowCount)
```

9.2.3 system.alert.queryAlertStatus

Description

Queries the alerting system for the current status of all alerts. By default, flatten mode is on, which means that you will get a single entry per alert path. If you turn flatten off, you'll get a row for *each state* of the alert. This can be important for alerts that have overlapping states.

The results of this function are a dataset with the following columns:
System - The system that issued the alert.
Path - The path to the alert item
Display Path - The custom display path (if any) for the alert. Will be the Path if no Display Path is configured.
State Name - The state name for the alert. If flatten is true, this will be the highest severity active alert state. If no state is active, this will be the most recently cleared alert state.
Severity - The severity, as a string.
Severity Code - The severity as an integer. 0-4, low-high.
Active - A boolean indicating whether this alert state is currently active.
Active Timestamp - The time at which this alert went active. May be null.
Active Value - The value that triggered this alert to go active.
Cleared - A boolean indicating whether this alert is currently clear.
Cleared Timestamp - The time at which this alert cleared. May be null.
Cleared Value - The value that cleared the alert.
Acked - A boolean indicating whether or not this alert has been acknowledged.
Ack Timestamp - The time that the alert was acknowledged. May be null.
Ack user - The user who acknowledged the alert.
Notes - The notes field for the alert
Flags - A bitmask representing the current alert state. 0x01= Active, 0x02=Cleared, 0x04=Acknowledged. So if the alert is active and acknowledged, but not cleared, this will be 0x01 | 0x04 = 5;

This function accepts *keyword-style* invocation. See also: Functions / Keyword Invocation

Syntax

system.alert.**queryAlertStatus(**system, path, stateName, minSeverity, maxSeverity, activeAndUnacked, activeAndAcked, clearAndUnacked, clearAndAcked, flatten, displayPath**)**

Parameters
String `system` - Filter string to restrict results based on the alert system.
String `path` - Filter string to restrict results based on the alert path
String `stateName` - Filter string to restrict results based on the alert state name
Integer `minSeverity` - Minimum severity to return. Defaults to 0 (Low).
Integer `maxSeverity` - Maximum severity to return. Defaults to 4 (High).
Boolean `activeAndUnacked` - Whether or not to return alerts that are currently active and unacknowledged. Default is true.

© 2011 Inductive Automation

Boolean `activeAndAcked` - Whether or not to return alerts that are currently active and have been acknowledged. Default is false.
Boolean `clearAndUnacked` - Whether or not to return alerts that are cleared and unacknowledged. Default is false.
Boolean `clearAndAcked` - Whether or not to return alerts that are cleared and have been acknowledged. Default is false.
Boolean `flatten` - If true, will flatten results so that there is only one entry per alert path, matching the highest active state. Default is true.
String `displayPath` - Filter string to restrict results based on the alert's display path

Returns
Dataset - A dataset containing the alerts in the system that match the filters.

Scope
All

Examples

This script will query the alert status for currently active alerts and push the results into a table.

```
results = system.alert.queryAlertStatus(flatten=1,activeAndUnacked=1, activeAndAcked=1)
event.source.parent.getComponent("Table").data=results
```

This *expression binding* will return the count of currently active alerts with a severity of Medium or higher, checking once a second.

```
runScript(
    "system.alert.queryAlertStatus(activeAndAcked=1, minSeverity=2).rowCount",
    1000
)
```

9.3 system.dataset

9.3.1 system.dataset.addRow

Description

Takes a dataset and returns a new dataset with a new row added or inserted into it. Datasets are immutable, so it is important to realize that this function does not actually add a row to a dataset. You'll need to do something with the new dataset that this function creates to achieve something useful. If the `rowIndex` argument is omitted, the row will be appended to the end of the dataset.

Syntax

```
system.dataset.addRow(dataset [, rowIndex], row)
```

Parameters
Dataset `dataset` - The starting dataset. Please be aware that this dataset will not actually be modified (datasets are immutable), but rather will be the starting point for creating a new dataset.
int `rowIndex` - The index (starting at 0) at which to insert the new row. Will throw an IndexError if less than zero or greater than the length of the dataset. If omitted, the new row will be appended to the end. [optional]
PySequence `row` - A Python sequence representing the data for the new row. Its length must equal the number of columns in the dataset.

Returns
Dataset - A new dataset with the new row inserted or appended.

Scope

© 2011 Inductive Automation

All

Examples

This example would add a new option to a Dropdown List by adding a row to its underlying dataset. Notice how the last line *assigns* the return value of the `addRow` function to the dropdown's `data` property.

```
dropdown = event.source.parent.getComponent("Dropdown")
newRow = [5, "New Option"]
dropdown.data = system.dataset.addRow(dropdown.data, newRow)
```

This snippet would add a new option into a Dropdown component just like above, but at the beginning:

```
dropdown = event.source.parent.getComponent("Dropdown")
newRow = [5, "New Option"]
dropdown.data = system.dataset.addRow(dropdown.data, 0, newRow)
```

9.3.2 system.dataset.dataSetToExcel

Description

Formats the contents of one or more datasets as an excel spreadsheet, returning the results as a string. Each dataset specified will be added as a worksheet in the Excel workbook. This function uses an xml-format for Excel spreadsheets, not the native Excel file format.

Syntax

```
system.dataset.dataSetToExcel(showHeaders, datasets)
```

Parameters

boolean `showHeaders` - If true (1), the spreadsheet will include a header row.

Object[] `datasets` - A sequence of datasets, one for each sheet in the resulting workbook.

Returns

String - An Excel-compatible XML-based workbook, with one worksheet per dataset.

Scope

All

Examples

This snippet would run a SQL query against a database, and turn the results into a string that is XML that Excel can open. It then writes the string to a file on the local hard drive.

```
results = system.db.runQuery("SELECT * FROM example1 LIMIT 100")
results = system.dataset.toDataSet(results)
spreadsheet = system.dataset.dataSetToExcel(1, [results])
filePath = "C:\\output\\results.xls"
system.file.writeFile(filePath, spreadsheet)
```

See also:
system.dataset.exportExcel

9.3.3 system.dataset.dataSetToHTML

Description

Formats the contents of a dataset as an HTML page, returning the results as a string. Uses the <table> element to create a data table page.

© 2011 Inductive Automation

Syntax

```
system.dataset.dataSetToHTML(showHeaders, dataset, title)
```

Parameters

boolean `showHeaders` - If true(1), the HTML table will include a header row.
Dataset `dataset` - The dataset to export
String `title` - The title for the HTML page.

Returns

String - The HTML page as a string.

Scope

All

Examples

This snippet would run a SQL query against a database, and turn the results into a string containing HTML. It then writes the string to a file on the local hard drive.

```
results = system.db.runQuery("SELECT * FROM example1 LIMIT 100")
results = system.dataset.toDataSet(results)
html = system.dataset.dataSetToHTML(1, results, "Production Report")
filePath = "C:\\output\\results.html"
system.file.writeFile(filePath, html)
```

See also:
system.dataset.exportHTML

9.3.4 system.dataset.deleteRow

Description

Takes a dataset and returns a new dataset with a row removed. Datasets are immutable, so it is important to realize that this function does not actually remove the row from the argument dataset. You'll need to do something with the new dataset that this function creates to achieve something useful.

Syntax

```
system.dataset.deleteRow(dataset, rowIndex)
```

Parameters

Dataset `dataset` - The starting dataset. Please be aware that this dataset will not actually be modified (datasets are immutable), but rather will be the starting point for creating a new dataset.
int `rowIndex` - The index (starting at 0) of the row to delete. Will throw an IndexError if less than zero or greater than len(dataset)-1.

Returns

Dataset - A new dataset with the specified row removed.

Scope

All

Examples

This example would remove the selected row from a List component, by re-assigning the List's `data` property to the new dataset returned by the `deleteRow` function.

© 2011 Inductive Automation

```
myList = event.source.parent.getComponent("List")
row = myList.selectedIndex
if row != -1: # make sure there is something selected
   myList.data = system.dataset.deleteRow(myList.data, row)
```

9.3.5 system.dataset.exportCSV

Description

Exports the contents of a dataset as a CSV file, prompting the user to save the file to disk.

Syntax

system.dataset.**exportCSV(**filename, showHeaders, dataset**)**

Parameters

String `filename` - A suggested filename to save as.
boolean `showHeaders` - If true (1), the CSV file will include a header row.
Dataset `dataset` - The dataset to export.

Returns

String - The path to the saved file, or None if the action was canceled by the user.

Scope

Client

Examples

This snippet would prompt the user to save the data currently displayed in a Table component to a CSV file, and would open the file (in an external program, presumably Excel) after a successful save.

```
table = event.source.parent.getComponent("Table")
filePath = system.dataset.exportCSV("data.csv", 1, table.data)
if filePath != None:
   system.net.openURL("file://"+filePath)
```

See also:
system.dataset.dataSetToCSV

9.3.6 system.dataset.exportExcel

Description

Exports the contents of a dataset as an Excel spreadsheet, prompting the user to save the file to disk. Uses the same format as the `dataSetToExcel` function.

Syntax

system.dataset.**exportExcel(**filename, showHeaders, dataset**)**

Parameters

String `filename` - A suggested filename to save as.
boolean `showHeaders` - If true (1), the spreadsheet will include a header row.
Object[] `dataset` - A sequence of datasets, one for each sheet in the resulting workbook.

Returns

String - The path to the saved file, or None if the action was canceled by the user.

Scope

© 2011 Inductive Automation

Client

Examples

This snippet would prompt the user to save the data currently displayed in a Table component to an Excel-compatible spreadsheet file, and would open the file after a successful save.

```
table = event.source.parent.getComponent("Table")
filePath = system.dataset.exportExcel("data.xls", 1, table.data)
if filePath != None:
   system.net.openURL("file://"+filePath)
```

See also:
system.dataset.dataSetToExcel

9.3.7 system.dataset.exportHTML

Description

Exports the contents of a dataset to an HTML page. Prompts the user to save the file to disk.

Syntax

```
system.dataset.exportHTML(filename, showHeaders, dataset, title)
```

Parameters

String `filename` - A suggested filename to save as.
boolean `showHeaders` - If true (1), the HTML tabl will include a header row.
Dataset `dataset` - The dataset to export.
String `title` - The title for the HTML page.

Returns

String - The path to the saved file, or None if the action was canceled by the user.

Scope

Client

Examples

This snippet would prompt the user to save the data currently displayed in a Table component to an HTML file, and would open the file in the default web browser after a successful save.

```
table = event.source.parent.getComponent("Table")
filePath = system.dataset.exportHTML("data.html", 1, table.data,"Production Report")
if filePath != None:
   system.net.openURL("file://"+filePath)
```

See also:
system.dataset.exportHTML

9.3.8 system.dataset.fromCSV

Syntax

```
system.dataset.fromCSV(csv)
```

Parameters

String `csv` - A string holding a CSV dataset.

© 2011 Inductive Automation

Returns
Dataset - A new dataset.

Scope
All

9.3.9 system.dataset.setValue

Description

Takes a dataset and returns a new dataset with a one value altered. Datasets are immutable, so it is important to realize that this function does not actually set a value in the argument dataset. You'll need to do something with the new dataset that this function creates to achieve something useful.

Syntax

system.dataset.**setValue(**dataset, rowIndex, columnName, value**)**

Parameters
Dataset `dataset` - The starting dataset. Will not be modified (datasets are immutable), but acts as the basis for the returned dataset.
int `rowIndex` - The index of the row to set the value at (starting at 0)
String `columnName` - The name of the column to set the value at. Case insensitive.
PyObject `value` - The new value for the specified row/column.

Returns
Dataset - A new dataset, with the new value set at the given location.

Scope
All

system.dataset.**setValue(**dataset, rowIndex, columnIndex, value**)**

Parameters
Dataset `dataset` - The starting dataset. Will not be modified (datasets are immutable), but acts as the basis for the returned dataset.
int `rowIndex` - The index of the row to set the value at (starting at 0)
int `columnIndex` - The index of the column to set the value at (starting at 0)
PyObject `value` - The new value for the specified row/column.

Returns
Dataset - A new dataset, with the new value set at the given location.

Scope
All

Examples

This snippet could be used for a Button's `actionPerformed` event to change the selected cell's value in a Table component to zero.

```
table = event.source.parent.getComponent("Table")
selRow = table.getSelectedRow()
selCol = table.getSelectedColumn()
if selRow != -1 and selCol != -1:
   newData = system.dataset.setValue(table.data, selRow, selCol, 0.0)
   table.data = newData
```

© 2011 Inductive Automation

9.3.10 system.dataset.toCSV

Syntax

```
system.dataset.toCSV(dataset, showHeaders, forExport)
```

Parameters

Dataset `dataset` - The dataset to export to CSV.

Boolean `showHeaders` - If set to true(1), a header row will be present in the CSV. Default is true(1).

Boolean `forExport` - If set to true(1), extra header information will be present in the CSV data which is necessary for the CSV to be compatible with the fromCSV method. Overrides showHeaders. Default is false(0).

Returns

String - The CSV data as a string

Scope

All

9.3.11 system.dataset.toDataSet

Description

This function is used to 1) convert PyDataSets to DataSets, and 2) create new datasets from raw Python lists. See also: Working with Datatypes / Datasets.

Syntax

```
system.dataset.toDataSet(dataset)
```

Parameters

PyDataSet `dataset` - A PyDataSet object to convert.

Returns

Dataset - The newly created dataset.

Scope

All

```
system.dataset.toDataSet(headers, data)
```

Parameters

PySequence `headers` - The column names for the dataset to create.

PySequence `data` - A list of rows for the new dataset. Each row must have the same length as the headers list, and each value in a column must be the same type.

Returns

Dataset - The newly created dataset.

Scope

All

Examples

This first example shows how this function can be used to convert from a PyDataSet (which is what system.db.runQuery returns) to a normal DataSet, which is the datatype of a Table component's `data` property.

```
pyDataSet = system.db.runQuery("SELECT * FROM example1 LIMIT 100")
```

© 2011 Inductive Automation

```
table = event.source.parent.getComponent("Table")
normalDataSet = system.dataset.toDataSet(pyDataSet)
table.data = normalDataSet
```

This second example shows how to use this function to create a new dataset out of a python sequence that you have filled in. In this case, the sequence is created via a for loop appending rows to a list.

```
#  Generate the Rows
rows = []
for x in range(10):
   oneRow = ["Row %d" % x, x+15]
   rows.append(oneRow)

# Generate the DataSet
headers = ["RowID", "Value"]
data = system.dataset.toDataSet(headers, rows)

# Use our new dataset to fill in a Table
table = event.source.parent.getComponent("Table")
table.data = data
```

9.3.12 system.dataset.toPyDataSet

Description

This function converts from a normal DataSet to a PyDataSet, which is a wrapper class which makes working with datasets more Python-esque. See also: Working with Datatypes / Datasets.

Syntax

system.dataset.**toPyDataSet(**dataset**)**

Parameters

Dataset `dataset` - A DataSet object to convert into a PyDataSet.

Returns

PyDataSet - The newly created PyDataSet.

Scope

All

Examples

This example script would be added to a button that is in the same container as the table you are working with. It grabs the data of the Table component, and adds up the values in the column named "Value", displaying the result to the user.

```
# Get a Table component's data
table = event.source.parent.getComponent("Table")
data = system.dataset.toPyDataSet(table.data)

# Loop through the data, summing the Value column
value = 0.0
for row in data:
   value += row["Value"]

# Show the user the sum of the Value column
system.gui.messageBox("The value is: %f" % value)
```

© 2011 Inductive Automation

9.3.13 system.dataset.updateRow

Description

Takes a dataset and returns a new dataset with a one row altered. Datasets are immutable, so it is important to realize that this function does not actually change the row in the argument dataset. You'll need to do something with the new dataset that this function creates to achieve something useful.

To alter the row, this function takes a Python dictionary to represent the changes to make to the specified row. The keys in the dictionary are used to find the columns to alter. See also: Sequences and Dictionaries.

Syntax

```
system.dataset.updateRow(dataset, rowIndex, changes)
```

Parameters

Dataset `dataset` - The starting dataset. Will not be modified (datasets are immutable), but acts as the basis for the returned dataset.

int `rowIndex` - The index of the row to update (starting at 0)

PyDictionary `changes` - A Dictionary of changes to make. They keys in the dictionary should match column names in the dataset, and their values will be used to update the row.

Returns

Dataset - A new dataset with the values at the specified row updated according to the values in the dictionary.

Scope

All

Examples

This example could be used to dynamically change the data that an Easy Chart displays. In this simple example, we assume that the chart is always configured to display a single tank's level. This script would update the pen being displayed using a dynamic tank number.

```
# Generate new tag name and tag path
tankNumber = 5
newName = "Tank%d Level" % tankNumber
newPath = "Tanks/Tank%d/Level" % tankNumber

# Consolidate changes into a dictionary
updates = {"NAME": newName, "TAG_PATH":newPath}

# Update the Easy Chart
chart = event.source.parent.getComponent("Easy Chart")
newPens = system.dataset.updateRow(chart.tagPens, 0, updates)
chart.tagPens = newPens
```

9.4 system.db

9.4.1 system.db.beginTransaction

Description

Begins a new database transaction. Database transactions are used to execute multiple queries in an atomic fashion. After executing queries, you must either *commit* the transaction to have your

© 2011 Inductive Automation

changes take effect, or *rollback* the transaction which will make all operations since the last commit not take place. The transaction is given a new unique string code, which is then returned. You can then use this code as the `tx` argument for other `system.db.*` function calls to execute various types of queries using this transaction.

An open transaction consumes one database connection until it is closed. Because leaving connections open indefinitely would exhaust the connection pool, each transaction is given a timeout. Each time the transaction is used, the timeout timer is reset. For example, if you make a transaction with a timeout of one minute, you must use that transaction at least once a minute. If a transaction is detected to have timed out, it will be automatically closed and its transaction id will no longer be valid.

Syntax

```
system.db.beginTransaction(database, isolationLevel, timeout)
```

Parameters

String `database` - The name of the database connection to create a transaction in. Use "" for the project's default connection.

Integer `isolationLevel` - The transaction isolation level to use. Use one of the four constants: system.db.READ_COMMITTED, system.db.READ_UNCOMMITTED, system.db.REPEATABLE_READ, or system.db.SERIALIZABLE

Long `timeout` - The amount of time, in milliseconds, that this connection is allowed to remain open without being used. Timeout counter is reset any time a query or call is executed against the transaction, or when committed or rolled-back.

Returns

String - The new transaction ID. You'll use this ID as the "tx" argument for all other calls to have them execute against this transaction.

Scope

All

Examples

This example would start a transaction with a 5 second timeout against the project's default database, using the default isolation level. Then it executes a series of update calls, and commits and closes the transaction.

```
txId = system.db.beginTransaction(timeout=5000)
status=2

for machineId in range(8):
   system.db.runPrepUpdate("UPDATE MachineStatus SET status=? WHERE ID=?", args=[status, machin

system.db.commitTransaction(txId)
system.db.closeTransaction(txId)
```

9.4.2 system.db.closeTransaction

Description

Closes the transaction with the given ID. Note that you must commit or rollback the transaction before you close it. Closing the transaction will return it's database connection to the pool. The transaction ID will no longer be valid.

Syntax

© 2011 Inductive Automation

```
system.db.closeTransaction(tx)
```

Parameters
String `tx` - The transaction ID.

Returns
nothing

Scope
All

9.4.3 system.db.commitTransaction

Description

Performs a commit for the given transaction. This will make all statements executed against the transaction since its beginning or since the last commit or rollback take effect in the database. Until you commit a transaction, any changes that the transaction makes will not be visible to other connections. Note that if you are done with the transaction, you must close it afterward you commit it.

Syntax

```
system.db.commitTransaction(tx)
```

Parameters
String `tx` - The transaction ID.

Returns
nothing

Scope
All

9.4.4 system.db.createSProcCall

Description

Creates an **SProcCall** object, which is a stored procedure *call context*. This is an object that is used to configure a call to a stored procedure. Once configured, you'd use system.db.execSProcCall to call the stored procedure. The call context object then holds any results from the stored procedure.

The SProcCall object has the following functions used for registering parameters:

`SPRocCall.registerInParam`(`index` *OR* `name`, `typeCode`, `value`)
`SPRocCall.registerOutParam`(`index` *OR* `name`, `typeCode`)
`SPRocCall.registerReturnParam`(`typeCode`)

These functions are used to register any in/out parameters for the stored procedure. Parameters can be referenced by index (starting at 1, not 0), or by name. To register an in/out parameter, you simply register it twice - once as an input parameter with the value you'd like to pass to the stored procedure, and once as an output parameter. N.B. not all JDBC drivers support named procedure parameters.

If your function returns a value, you must use `registerReturnParam` to specify the datatype of the returned value. Note that this is different from stored procedures that return a *result set*, which doesn't require any setup on the SProcCall object. Some database systems call stored procedures

© 2011 Inductive Automation

that return a value "functions" instead of "procedures".

For all of these functions, you'll need to specify a *type code*. These are codes defined by the JDBC specification. For your convenience, the codes exist as constants in the `system.db` namespace. Each type code will be mapped to a database-specific type by the JDBC driver. Not all type codes will be recognized by all JDBC drivers. The following type code constants are available:

BIT	REAL	LOGVARCHAR	LONGVARBINARY	BLOB	NCHAR
TINYINT	DOUBLE	DATE	NULL	CLOB	NVARCHAR
SMALLINT	NUMERIC	TIME	OTHER	REF	LONGNVARCHAR
INTEGER	DECIMAL	TIMESTAMP	DISTINCT	DATALINK	NCLOB
BIGINT	CHAR	BINARY	STRUCT	BOOLEAN	SQLXML
FLOAT	VARCHAR	VARBINARY	ARRAY	ROWID	ORACLE_CURSOR

Once the call context has been executed, you can retrieve the result set, return value, and output parameter values (if applicable) by calling the following functions:

`SProcCall.getResultSet`() - returns a dataset that is the resulting data of the stored procedure, if any.
`SProcCall.getUpdateCount`() - returns the number of rows modified by the stored procedure, or -1 if not applicable.
`SProcCall.getReturnValue`() - returns the return value, if registerReturnParam had been called.
`SProcCall.getOutParamValue`(`index` *OR* `name`) - returns the value of the previously registered out-parameter.

Syntax

```
system.db.createSProcCall(procedureName, database, tx)
```

Parameters

String `procedureName` - The named of the stored procedure to call.
String `database` - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.
String `tx` - A transaction identifier. If omitted, the call will be executed in its own transaction.

Returns

SProcCall - A stored procedure call context, which can be configured and then used as the argument to system.db.execSProcCall.

Scope

All

Examples

This example would call a stored procedure named "start_batch" against the current project's default database connection that had no input or output parameters, and did not return any values or results:

```
call = system.db.createSProcCall("start_batch")
system.db.execSProcCall(call)
```

This example would call a stored procedure "get_shift_workers" with no arguments, which returned a result set of employees for the current shift. It then pushes the resulting dataset into a Table component:

© 2011 Inductive Automation

```
call = system.db.createSProcCall("get_shift_workers")
system.db.execSProcCall(call)

results = call.getResultSet()
table = event.source.parent.getComponent("Table")
table.data = results
```

This example would call a stored procedure that took two arguments, the first an integer and the second a string. It also is configured to return an integer value.

```
call = system.db.createSProcCall("perform_calculation")
call.registerReturnParam(system.db.INTEGER)
call.registerInParam(1, system.db.INTEGER, 42)
call.registerInParam(2, system.db.VARCHAR, "DC-MODE")

system.db.execSProcCall(call)

#Print the result to the console
print call.getReturnValue()
```

This example would do the same as the one above, except for a stored procedure that returned its value using an out-parameter. It also uses named argument names instead of indexed arguments.

```
call = system.db.createSProcCall("perform_calculation")
call.registerInParam("arg_one", system.db.INTEGER, 42)
call.registerInParam("arg_two", system.db.VARCHAR, "DC-MODE")
call.registerOutParam("output_arg", system.db.INTEGER)

system.db.execSProcCall(call)

#Print the result to the console
print call.getOutParamValue("output_arg")
```

9.4.5 system.db.dateFormat

Description

This function is used to format Dates nicely as strings. It uses a *format string* to guide its formatting behavior. Learn more about date formatting in Working with Datatypes / Dates

Expert Tip: This function uses the Java class java.text.SimpleDateFormat internally, and will accept any valid format string for that class.

Syntax

```
system.db.dateFormat(date, formatPattern)
```

Parameters

Date `date` - The Date object that you'd like to format

String `formatPattern` - A format pattern string to apply.

Returns

String - The date as a string formatted according to hte format pattern.

Scope

All

© 2011 Inductive Automation

Examples

This example will display a message box on a button press that displays the selected date (without the time) from a Calendar component, in a format like "Feb 3, 2009"

```
date = event.source.parent.getComponent("Calendar").latchedDate
toDisplay = system.db.dateFormat(date, "MMM d, yyyy")
system.gui.messageBox("The date you selected is: %s" % toDisplay)
```

This example would do the same as the one above, but also display the time, in a format like: "Feb 3, 2009 8:01pm"

```
date = event.source.parent.getComponent("Calendar").latchedDate
toDisplay = system.db.dateFormat(date, "MMM d, yyyy")
system.gui.messageBox("The date you selected is: %s" % toDisplay)
```

This example would take two dates from two Popup Calendar components, format them in a manner that the database understands, and then use them in a SQL query to limit the results to a certain date range.

```
startDate = event.source.parent.getComponent("StartDate").date
endDate = event.source.parent.getComponent("EndDate").date
startDate = system.db.dateFormat(startDate, "yyyy-MM-dd HH:mm:ss")
endDate = system.db.dateFormat(endDate, "yyyy-MM-dd HH:mm:ss")
query = "SELECT * FROM mytable WHERE t_stamp >= '%s' AND t_stamp <= '%s'" % (startDate, endDate
results = system.db.runQuery(query)
event.source.parent.getComponent("Table").data = results
```

9.4.6 system.db.execSProcCall

Description

Executes a stored procedure call. The one parameter to this function is an **SProcCall** - a stored procedure *call context*. See the description of system.db.createSProcCall for more information and examples.

Syntax

```
system.db.execSProcCall(callContext)
```

Parameters

SProcCall `callContext` - A stored procedure call context, with any input, output, and/or return value parameters correctly configured. Use system.db.createSProcCall to create a call context.

Returns

nothing

Scope

All

9.4.7 system.db.getConnectionInfo

Description

Returns a dataset of information about a single database connection, as specified by the `name` argument.

Syntax

```
system.db.getConnectionInfo(name)
```

© 2011 Inductive Automation

Parameters

String `name` - The name of the database connection to find information about

Returns

Dataset - A dataset containing information about the named database connection, or an empty dataset if the connection wasn't found.

Scope

All

9.4.8 system.db.getConnections

Description

Returns a dataset of information about each configured database connection. Each row represents a single connection.

Syntax

```
system.db.getConnections()
```

Parameters

none

Returns

Dataset - A dataset, where each row represents a database connection.

Scope

All

9.4.9 system.db.refresh

Description

This function will programmatically cause a SQL Query or DB Browse property binding to execute immediately. This is most often used for bindings that are set to *Polling - Off*. In this way, you cause a binding to execute on demand, when you know that the results of it's query will return a new result. To use it, you simply specify the component and name of the property on whose binding you'd like to refresh.

Syntax

```
system.db.refresh(component, propertyName)
```

Parameters

JComponent `component` - The component whose property you want to refresh

String `propertyName` - The name of the property that has a SQL Query binding that needs to be refreshed

Returns

boolean - True (1) if the property was found and refreshed successfully.

Scope

Client

Examples

This example could be placed in the actionPerformed event of a Button, to be used to refresh the data of a Table. Remember to use the *scripting name* of the property that you're trying to refresh, and that the property names are case-sensitive.

© 2011 Inductive Automation

```
table = event.source.parent.getComponent("Table")
system.db.refresh(table, "data")
```

9.4.10 system.db.rollbackTransaction

Description

Performs a rollback on the given connection. This will make all statements executed against this transaction since its beginning or since the last commit or rollback undone. Note that if you are done with the transaction, you must also close it afterward you do a rollback on it.

Syntax

```
system.db.rollbackTransaction(tx)
```

Parameters

String `tx` - The transaction ID.

Returns

nothing

Scope

All

9.4.11 system.db.runPrepQuery

Description

Runs a *prepared statement* against the database, returning the results in a PyDataSet.. Prepared statements differ from regular queries in that they can use a special placeholder, the question-mark character (`?`) in the query where any dynamic arguments would go, and then use an array of values to provide real information for those arguments. Make sure that the length of your argument array matches the number of question-mark placeholders in your query. This call should be used for SELECT queries.

This is a useful alternative to system.db.runQuery because it allows values in the WHERE clause, JOIN clause, and other clauses to be specified without having to turn those values into *strings*. This is safer because it protects against a problem known as a SQL injection attack, where a user can input data that affects the query's semantics.

Syntax

```
system.db.runPrepQuery(query, args, database, tx)
```

Parameters

String `query` - A query (typically a SELECT) to run as a prepared statement, with placeholders (?) denoting where the arguments go.

Object[] `args` - A list of arguments. Will be used in order to match each placeholder (?) found in the query.

String `database` - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

String `tx` - A transaction identifier. If omitted, the query will be executed in its own transaction.

Returns

PyDataSet

Scope

© 2011 Inductive Automation

All

Examples

This example would search for all records in a LogEntry table where the message contained a user-entered search term.

```
search = event.source.parent.getComponent("SearchFor").text
# Wrap the term in % signs for LIKE-style matching
search = '%' + search + '%'

results= system.db.runPrepQuery("SELECT * FROM LogEntry WHERE EntryText LIKE ?", [search])
event.source.parent.getComponent("Table").data = results
```

9.4.12 system.db.runPrepUpdate

Description

Runs a *prepared statement* against the database, returning the number of rows that were affected. Prepared statements differ from regular queries in that they can use a special placeholder, the question-mark character (`?`) in the query where any dynamic arguments would go, and then use an array of values to provide real information for those arguments. Make sure that the length of your argument array matches the number of question-mark placeholders in your query. This call should be used for UPDATE, INSERT, and DELETE queries.

This is extremely useful for two purposes:

1. This method avoids the problematic technique of concatenating user input inside of a query, which can lead to syntax errors, or worse, a nasty security problem called a SQL injection attack. For example, if you have a user-supplied string that is used in a WHERE clause, you use single-quotes to enclose the string to make the query valid. What happens in the user has a single-quote in their text? Your query will fail. Prepared statements are immune to this problem.
2. This is the only way to write an INSERT or UPDATE query that has binary or BLOB data. Using BLOBs can be very hand for storing images or reports in the database, where all clients have access to them.

Syntax

```
system.db.runPrepUpdate(query, args, database, tx, getKey)
```

Parameters

String `query` - A query (typically an UPDATE, INSERT, or DELETE) to run as a prepared statement, with placeholders (?) denoting where the arguments go.

Object[] `args` - A list of arguments. Will be used in order to match each placeholder (?) found in the query.

String `database` - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

String `tx` - A transaction identifier. If omitted, the update will be executed in its own transaction.

Boolean `getKey` - A flag indicating whether or not the result should be the number of rows returned (getKey=0) or the newly generated key value that was created as a result of the update (getKey=1). Not all databases support automatic retrieval of generated keys.

Returns

Integer - The results of the query as a PyDataSet

Scope

© 2011 Inductive Automation

All

Examples

This example would gather some user entered text and insert it into the database.

```
userText = event.source.parent.getComponent("TextArea").text
userName = system.security.getUsername()
system.db.runPrepUpdate("INSERT INTO Comments (Name, UserComment) VALUES (?,?)", [userName, use
```

This code would read a file and upload it to the database

```
filename = system.file.openFile() # Ask the user to open a file
if filename != None:
   filedata = system.file.readFileAsBytes(filename)
   system.db.runPrepUpdate("INSERT INTO Files (file_data) VALUES (?)", [filedata])
```

This example inserts a new user and gives it the 'admin' role. Demonstrates the ability to retrieve a newly created key value.

```
#get the username/password
name = event.source.parent.getComponent('Name').text
desc = event.source.parent.getComponent('Description').text
building = event.source.parent.getComponent('Building').selectedValue

#insert the value
id = system.db.runPrepUpdate("INSERT INTO machines (machine_name, description) VALUES (?, ?)",

#add a row to the user role mapping table
system.db.runPrepUpdate("INSERT INTO machine_building_mapping (machine_id, building) VALUES (?,
```

9.4.13 system.db.runQuery

Description

Runs a SQL query, usually a SELECT query, against a database, returning the results as a dataset. If no database is specified, or the database is the empty-string "", then the current project's default database connection will be used. The results are returned as a PyDataSet, which is a wrapper around the standard dataset that is convenient for scripting. See also: Working with Datatypes / Datasets.

Syntax

```
system.db.runQuery(query, database, tx)
```

Parameters

String `query` - A SQL query, usually a SELECT query, to run.
String `database` - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.
String `tx` - A transaction identifier. If omitted, the query will be executed in its own transaction.

Returns

PyDataSet - The results of the query as a PyDataSet.

Scope

All

Examples

© 2011 Inductive Automation

Assuming the following dataset:

	Value
	3.55
	67.2
	9.87

If you executed the following code:

```
table = system.db.runQuery("SELECT * FROM TEST")
```

then `table[2]` would access the third row (rows are zero-indexed), and both `table[2][0]` and `table[2]["ID"]` would access the ID value of the third row. As further example of how to use the results of runQuery, here are seven different ways to print out the table, and their results follow. Note that some of the later methods exercise some more advanced Jython concepts such as list comprehensions and string formatting, but their intent should be obvious. Generally speaking, the more concise Jython code becomes, the more readable it is.

```
table = system.db.runQuery("SELECT * FROM Test")

print "Printing TEST Method 1..."
for row in table:
   for col in row:
      print col,
   print ""
print ""

print "Printing TEST Method 2..."
for row in table:
   print row[0], row[1]
print ""

print "Printing TEST Method 3..."
for row in table:
   print row["ID"], row["VALUE"]
print ""

print "Printing TEST Method 4..."
for rowIdx in range(len(table)):
   print "Row ",str(rowIdx)+": ", table[rowIdx][0], table[rowIdx][1]
print ""

print "Printing TEST Method 5..."
print [str(row[0])+", "+ str(row[1]) for row in table]
print ""

print "Printing TEST Method 6..."
print ["%s, %s" % (row["ID"],row["VALUE"]) for row in table]
print ""

print "Printing TEST Method 7..."
print [[col for col in row] for row in table]
print ""
```

© 2011 Inductive Automation

The results printed out would be:

```
Printing TEST Method 1...
0 3.55
1 67.2
2 9.87

Printing TEST Method 2...
0 3.55
1 67.2
2 9.87

Printing TEST Method 3...
0 3.55
1 67.2
2 9.87

Printing TEST Method 4...
Row 0: 0 3.55
Row 1: 1 67.2
Row 2: 2 9.87

Printing TEST Method 5...
['0, 3.55', '1, 67.2', '2, 9.87']

Printing TEST Method 6...
['0, 3.55', '1, 67.2', '2, 9.87']

Printing TEST Method 7...
[[0, 3.55], [1, 67.2], [2, 9.87]]
```

9.4.14 system.db.runScalarQuery

Description

Runs a query against a database connection just like the `runQuery` function, but only returns the value from the first row and column. If no results are returned from the query, the special value `None` is returned.

Syntax

system.db.**runScalarQuery(**query, database, tx**)**

Parameters

String `query` - A SQL query that should be designed to return one row and one column.
String `database` - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.
String `tx` - A transaction identifier. If omitted, the query will be executed in its own transaction.

Returns

Object - The value from the first row and first column of the results. Returns None if no rows were returned.

Scope

All

Examples

Example 1:

This code would count the number of active alarms, and acknowledge them all if there is at least one.

```
numAlarms = system.db.runScalarQuery("SELECT COUNT(*) FROM alarmstatus WHERE unacknowledged = 1
if numAlarms > 0:
   # There are alarms - acknowledge all of them
   system.db.runUpdateQuery("UPDATE alarmstatus SET unacknowledged = 0")
```

Example 2:

This code would read a single value from a table and show it to the user an a popup box.

```
lakeLevel = system.db.runScalarQuery("SELECT Level FROM LakeInfo WHERE LakeId='Tahoe'")
system.gui.messageBox("The lake level is: %d feet" % lakeLevel)
```

9.4.15 system.db.runUpdateQuery

Description

Runs a query against a database connection, returning the number of rows affected. Typically this is an UPDATE, INSERT, or DELETE query. If no database is specified, or the database is the empty-string "", then the current project's default database connection will be used.

Note that you may want to use the `runPrepUpdate` query if your query is constructed with user input (to avoid the user's input from breaking your syntax) or if you need to insert binary or BLOB data.

Syntax

```
system.db.runUpdateQuery(query, database, tx, getKey)
```

Parameters

String `query` - A SQL query, usually an INSERT, UPDATE, or DELETE query, to run.

String `database` - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

String `tx` - A transaction identifier. If omitted, the update will be executed in its own transaction.

Boolean `getKey` - A flag indicating whether or not the result should be the number of rows returned (getKey=0) or the newly generated key value that was created as a result of the update (getKey=1). Not all databases support automatic retrieval of generated keys.

Returns

Integer - The number of rows affected by the query, or the key value that was generated, depending on the value of the getKey flag.

Scope

All

Examples

This code would acknowledge all unacknowledged alarms # and show the user how many alarms were acknowledged.

© 2011 Inductive Automation

```
rowsChanged = system.db.runUpdateQuery("UPDATE alarmstatus SET unacknowledged = 0")
system.gui.messageBox("Acknowledged %d alarms" % rowsChanged)
```

This code would insert a new recipe step into a recipe table, after asking the user how many gallons of syrup should be added on this recipe step.

```
inputText = system.db.inputBox("How many gallons?", "12.3")
# Make sure the user didn't hit cancel
if inputText != None:
   # Make sure the input is a number
   gallons = float(inputText)
   # Detect the next step number by adding 1 to the last step number
   nextStepNum = system.db.runScalarQuery("SELECT MAX(StepNum) + 1 FROM RecipeSteps")
   # Insert recipe step
   system.db.runUpdateQuery("INSERT INTO RecipeSteps (StepNum, Gallons) VALUES (%d, %f)" % (nex
```

This example inserts a new user and gives it the 'admin' role. Demonstrates the ability to retrieve a newly created key value.

```
#get the username/password
name = event.source.parent.getComponent('Name').text
desc = event.source.parent.getComponent('Description').text
building = event.source.parent.getComponent('Building').selectedValue

#insert the value
id = system.db.runUpdateQuery("INSERT INTO machines (machine_name, description) VALUES ('%s', '

#add a row to the user role mapping table
system.db.runUpdateQuery("INSERT INTO machine_building_mapping (machine_id, building) VALUES (%
```

9.5 system.file

9.5.1 system.file.fileExists

Description

Checks to see if a file at a given path exists.

Syntax

```
system.file.fileExists(filepath)
```

Parameters

String `filepath` - The path of the file to check.

Returns

boolean - True (1) if the file exists, false (0) otherwise.

Scope

All

Examples

This basic example shows how the `fileExists` function is used in its simplest form:

```
if system.file.fileExists("C:\\temp_file.txt"):
   system.gui.messageBox("Yes, the file exists")
else:
```

© 2011 Inductive Automation

```
   system.gui.messageBox("No, it doesn't exist")
```

This code uses the `fileExists` function, along with other `system.file.*` functions, to prompt the user to confirm that they want to overwrite an existing file.

```
filename = system.file.saveFile(name)
if filename != None:
   reallyWrite = 1
   if system.file.fileExists(filename):
      reallyWrite = system.gui.confirm("File '%s' already exists. Overwrite?" % filename)
   if reallyWrite:
      system.file.writeFile(filename, "This will be the contents of my new file")
```

9.5.2 system.file.getTempFile

Description

Creates a new temp file on the host machine with a certain extension, returning the path to the file. The file is marked to be removed when the Java VM exits.

Syntax

```
system.file.getTempFile(extension)
```

Parameters

String `extension` - An extension, like ".txt", to append to the end of the temporary file.

Returns

String - The path to the newly created temp file.

Scope

All

Examples

This code writes some data to a temorary file, and then opens that file. Assume that the data variable holds the contents of an excel (xls) file.

```
filename = system.file.getTempFile("xls")
system.file.writeFile(filename, data)
system.net.openURL("file://" + filename)
```

9.5.3 system.file.openFile

Description

Shows an "Open File" dialog box, prompting the user to choose a file to open. Returns the path to the file that the user chose, or `None` if the user canceled the dialog box. An extension can optionally be passed in that sets the filetype filter to that extension.

Syntax

```
system.file.openFile([extension])
```

Parameters

String `extension` - A file extension, like "pdf", to try to open. [optional]

Returns

String - the path to the selected file, or None if canceled.

Scope

© 2011 Inductive Automation

Client

Examples

This code would prompt the user to open a file of type 'gif'. If None is returned, it means the user canceled the open dialog box.

```
path = system.db.openFile('gif')
if path != None:
   # do something with the file
```

9.5.4 system.file.readFileAsBytes

Description

Opens the file found at path `filename`, and reads the entire file. Returns the file as an array of bytes. Commonly this array of bytes is uploaded to a database table with a column of type BLOB (Binary Large OBject). This upload would be done through an INSERT or UPDATE SQL statement run through the `system.db.runPrepUpdate` function. You could also write the bytes to another file using the `system.file.writeFile` function, or send the bytes as an email attachment using `system.net.sendEmail`.

Syntax

```
system.file.readFileAsBytes(filepath)
```

Parameters

String `filepath` - The path of the file to read.

Returns

byte[] - The contents of the file as an array of bytes.

Scope

All

Examples

This code would prompt the user to choose a file. If the user chooses a file, it would then read that file and upload it to a database table called Files into a BLOB column called file_data.

```
path = system.file.openFile()

if path != None:
   bytes = system.file.readFileAsBytes(filename)
   system.db.runPrepUpdate("INSERT INTO Files (file_data) VALUES (?)", (bytes))
```

9.5.5 system.file.readFileAsString

Description

Opens the file found at path `filename`, and reads the entire file. Returns the file as a string. Common things to do with this string would be to load it into the text property of a component, upload it to a database table, or save it to another file using `system.file.writeFile` function.

Syntax

```
system.file.readFileAsString(filepath)
```

© 2011 Inductive Automation

Parameters

String `filepath` - The path of the file to read.

Returns

String - The contents of the file as a string.

Scope

All

Examples

This code would prompt the user to choose a text file. If the user chooses a file, it would then set a text area on the screen to display the file.

```
path = system.file.openFile("txt")

if path != None:
   contents = system.file.readFileAsString(path)
   event.source.parent.getComponent("Text Area").text = contents
```

9.5.6 system.file.saveFile

Description

Prompts the user to save a new file named `filename`. The optional `extension` and `typeDesc` arguments will be used for a file type filter, if any. If the user accepts the save, the path to that file will be returned. If the user cancels the save, `None` will be returned.

Syntax

```
system.file.saveFile(filename [, extension] [, typeDesc])
```

Parameters

String `filename` - A file name to suggest to the user.

String `extension` - The appropriate file extension, like "jpeg", for the file. [optional]

String `typeDesc` - A description of the extension, ilke "JPEG Image" [optional]

Returns

String - The path to the file that the user decided to save to, or None if they canceled.

Scope

Client

Examples

This code would prompt the user to save the text in a text area to a file.

```
path = system.file.saveFile("myfile.txt")
if path != None:
   system.file.writeFile(path, event.source.parent.getComponent("Text Area").text)
```

9.5.7 system.file.writeFile

Description

Writes the given data to the file at file path `filename`. If the file exists, the `append` argument determines whether or not it is overwritten (the default) or appended to. The data argument can be either a string or an array of bytes (commonly retrieved from a BLOB in a database or read from

© 2011 Inductive Automation

another file using `system.file.readFileAsBytes`).

Syntax

system.file.**writeFile(**filepath, charData **[**, append**])**

Parameters

String `filepath` - The path of the file to write to.
String `charData` - The character content to write to the file.
boolean `append` - If true(1), the file will be appended to if it already exists. If false(0), the file will be overwritten if it exists. The default is false(0). [optional]

Returns

nothing

Scope

All

system.file.**writeFile(**filepath, data **[**, append**])**

Parameters

String `filepath` - The path of the file to write to.
byte[] `data` - The binary content to write to the file.
boolean `append` - If true(1), the file will be appended to if it already exists. If false(0), the file will be overwritten if it exists. The default is false(0). [optional]

Returns

nothing

Scope

All

Examples

Example 1:

This code would download a BLOB from a database and save it to a file.

```
resultset = system.db.runQuery("SELECT file_data FROM Files WHERE id=12")
   if len(rs) > 0: # if the query returned anything...
      data = rs[0][0] # grab the BLOB at the 0th row and 0th column
      filename = system.file.saveFile("MyDownloadedFile.xyz")
      if filename != None:
         system.file.writeFile(filename, data)
```

Example 2:

This code would write the contents of a text area to a file.

```
data = event.source.parent.getComponent("Text Area").text
filename = system.file.saveFile("MyDownloadedFile.txt")
if filename != None:
   system.file.writeFile(filename, data)
```

© 2011 Inductive Automation

9.6 system.gui

9.6.1 system.gui.chooseColor

Description

Prompts the user to pick a color using the default color-chooser dialog box.

Syntax

```
system.gui.chooseColor(initialColor [, dialogTitle])
```

Parameters

Color `initialColor` - A color to use as a starting point in the color choosing popup.
String `dialogTitle` - The title for the color choosing popup. Defaults to "Choose Color" [optional]

Returns

Color - The new color chosen by the user.

Scope

Client

Examples

This code would be placed in the actionPerformed event of a button, and would change the background color of the container the button was placed in.

```
parent = event.source.parent
newColor = system.gui.chooseColor(parent.background)
parent.background = newColor
```

9.6.2 system.gui.color

Description

Creates a new color object, either by parsing a string or by having the RGB[A] channels specified explicitly.

Syntax

```
system.gui.color(color)
```

Parameters

String `color` - A string that will be coerced into a color. Can accept many formats, such as "red" or "#FF0000" or "255,0,0"

Returns

Color - The newly created color.

Scope

Client

```
system.gui.color(red, green, blue [, alpha])
```

Parameters

int `red` - The red component of the color, an integer 0-255.
int `green` - The green component of the color, an integer 0-255.

© 2011 Inductive Automation

int `blue` - The blue component of the color, an integer 0-255.
int `alpha` - The alpha component of the color, an integer 0-255. [optional]

Returns
Color - The newly created color.

Scope
Client

Examples
This example changes the background color of a component to red.

```
myComponent = event.source
myComponent.background = fpmi.gui.color(255,0,0) # turn the component red
```

9.6.3 system.gui.confirm

Description
Displays a confirmation dialog box to the user with "Yes" and "No" options, and a custom message.

Syntax

```
system.gui.confirm(message [, title])
```

Parameters
String `message` - The message to show in the confirmation dialog.
String `title` - The title for the confirmation dialog. [optional]

Returns
boolean - True (1) if the user selected "Yes", false (0) if the user selected "No"

Scope
Client

Examples
By using the confirm function in an if statement, we can let the user confirm an action. In this case, we shut down the plaint if the user confirms it, otherwise, we don't do anything.

```
if system.gui.confirm("Are you sure you want to shutdown the plant?", "Really Shutdown?"):
   system.db.runUpdateQuery("UPDATE ControlTable SET Shutdown=1")
```

9.6.4 system.gui.convertPointToScreen

Description
Converts a pair of coordinates that are relative to the upper-left corner of some component to be relative to the upper-left corner of the entire screen.

Syntax

```
system.gui.convertPointToScreen(x, y, event)
```

Parameters
int `x` - The X-coordinate, relative to the component that fired the event.
int `y` - The Y-coordinate, relative to the component that fired the event.
EventObject `event` - An event object for a component event.

© 2011 Inductive Automation

Returns
PyTuple - A tuple of (x,y) in screen coordinates.

Scope
Client

Examples

This example will get the coordinates where the mouse is (from the corner of the monitor) and display them in a label.

```
#get the screen coordinates of the pointer and write them to a label
coords = system.gui.convertPointToScreen(event.x,event.y,event)
event.source.parent.getComponent('Label').text = "x: %s y: %s" %(coords[0], coords[1])
```

9.6.5 system.gui.createPopupMenu

Description

Creates a new popup menu, which can then be shown over a component on a mouse event.

To use this function, first create a Python sequence whose entries are strings, and another sequence whose entries are *function objects*. The strings will be the items that are displayed in your popup menu, and when an item is clicked, its corresponding function will be run. Your functions must accept an `event` object as an argument. See also: Functions

To show the popup menu, store the menu object that this function returns, and then call its `show(event)` function, where `event` is the event object for a `mousePressed` or `mouseReleased` event on the component you wish the popup menu to be shown on.

Best Practices. It is best to have the menu object created only once via an application specific library function. Then, call the `show(event)` function on *both* the `mousePressed` and `mouseReleased` events on your component. The reason for this is that different operating systems (Windows, Linux, MacOS) differ in when they like to show the popup menu. The `show(event)` function detects when the right time is to show itself, either on mouse press or release. See the examples for more.

Syntax

```
system.gui.createPopupMenu(itemsDict)
```

Parameters
PyDictionary `itemsDict` - A dictionary of String:Function keys to create the popup menu. You can create sub-menus by using a nested dictionary of the same type as a dictionary value.

Returns
JPopupMenu - The javax.swing.JPopupMenu that was created.

Scope
Client

```
system.gui.createPopupMenu(itemNames, itemFunctions)
```

Parameters
PySequence `itemNames` - A list of names to create popup menu items with.
PySequence `itemFunctions` - A list of functions to match up with the names.

© 2011 Inductive Automation

Returns

JPopupMenu - The javax.swing.JPopupMenu that was created.

Scope

Client

Examples

This first example is a very basic to demonstrate the fundamentals of making a popup menu. Put the following script in the `mouseReleased` event of a component. This will only work on Windows - continue on for cross-platform instructions.

```
def sayHello(event):
    import system
    system.gui.messageBox("Hello World")

menu = system.gui.createPopupMenu({"Click Me":sayHello})

menu.show(event)
```

Because of the different popup-trigger settings on different operating systems, the preceding code will probably fail on Linux or a Mac. The way around this is to do the same code in both the `mousePressed` and `mouseReleased` events. In order to avoid code duplication, you'll want to factor out the code into a project script module.

The following, more sophisticated example shows a popup menu being used to acknowledge alarms in an alarm table by right-clicking on the table, and choosing either to acknowledge the selected alarm or all alarms. You would put this script in a project script module called `app.util`:

```
def getAlarmPopup():
    import system,app

    # This function will be the "Acknowledge" entry in the popup menu
    def ack(event):
        import system,app
        table = event.source
        selRow = table.selectedRow
        if selRow == -1:
            system.gui.warningBox("No alarm selected")
        elif table.model.getValueAt(selRow, 0) == 0:
            # In my table, the first column is the alarm's unacknowledged bit.
            system.gui.warningBox("Alarm already acknowledged")
        else:
            desc = table.model.getValueAt(selRow, 1)
            path = table.model.getValueAt(selRow, 2)
            message = "<html>Are you sure you want to acknowledge<br>%s?" % desc
            if system.gui.confirm(message,"Confirm"):
                app.auth.ackAlarm(desc,path)
                table.setSelectedRow(-1)

    # This function will be the "Acknowledge All" entry in the popup menu
    def ackAll(event):
        import system,app
        if system.gui.confirm("Are you sure you want to acknowledge all alarms?","Confirm"):
            app.auth.ackAllAlarms(event)

    # Finally, create the actual popup menu and return it
    alarmPopup = system.gui.createPopupMenu(["Acknowledge Alarm", "Acknowledge All"],[ack, ackAl
    return alarmPopup
```

© 2011 Inductive Automation

Now you could simply put this code in the Table's `mousePressed` and `mouseReleased` events:

```
menu = app.util.getAlarmPopup()
menu.show(event)
```

9.6.6 system.gui.errorBox

Description

Displays an error-style message box to the user.

Syntax

```
system.gui.errorBox(message [, title])
```

Parameters

String `message` - The message to display in an error box.
String `title` - The title for the error box. [optional]

Returns

nothing

Scope

Client

Examples

Turn on compressor #12, but only if the user has the right credentials.

```
if 'Supervisor' in system.security.getRoles():
    system.db.runUpdateQuery("UPDATE CompressorControl SET running=1 WHERE compNum = 12")
else:
    system.gui.errorBox("Unable to turn on Compressor 12. You don't have proper security privil
```

See also:
system.gui.messageBox
system.gui.warningBox

9.6.7 system.gui.getOpenedWindowNames

Description

Finds all of the currently open windows, returning a tuple of their paths.

Syntax

```
system.gui.getOpenedWindowNames()
```

Parameters

none

Returns

PyTuple - A tuple of strings, representing the path of each window that is open.

Scope

Client

Examples

This example prints out into the console the full path for each opened window.

```
windows = system.gui.getOpenedWindowNames()
```

© 2011 Inductive Automation

```
print 'There are %d windows open' % len(windows)
for path in windows:
    print path
```

9.6.8 system.gui.getOpenedWindows

Description

Finds all of the currently open windows, returning a tuple of references to them.

Syntax

```
system.gui.getOpenedWindows()
```

Parameters

none

Returns

PyTuple - A tuple of the opened windows. Not their names, but the actual window objects themselves.

Scope

Client

Examples

This example prints out the path of each currently opened window to the console.

```
windows = system.gui.getOpenedWindows()
print 'There are %d windows open' % len(windows)
for window in windows:
    print window.getPath()
```

9.6.9 system.gui.getParentWindow

Description

Finds the parent (enclosing) window for the component that fired an event, returning a reference to it.

Syntax

```
system.gui.getParentWindow(event)
```

Parameters

EventObject `event` - A component event object.

Returns

PyObject - The window that contains the component that fired the event.

Scope

Client

Examples

Use this in an event script to change the window's title.

```
window = system.gui.getParentWindow(event)
window.title='This is a new title'
```

© 2011 Inductive Automation

9.6.10 system.gui.getSibling

Description

Given a component event object, looks up a sibling component. Shortcut for `event.source.parent.getComponent("siblingName")`. If no such sibling is found, the special value `None` is returned.

Syntax

```
system.gui.getSibling(event, name)
```

Parameters

EventObject `event` - A component event object.
String `name` - The name of the sibling component.

Returns

PyObject - The sibling component itself.

Scope

Client

Examples

This example will get it's sibling Text Field's text, and use it.

```
textField = system.gui.getSibling(event, 'TextField (1)')
if textField is None:
   system.gui.errorBox("There is no text field!")
else:
   system.gui.messageBox("You typed: %s" % textField.text)
```

9.6.11 system.gui.getWindow

Description

Finds a reference to an open window with the given name. Throws a `ValueError` if the named window is not open or not found.

Syntax

```
system.gui.getWindow(name)
```

Parameters

String `name` - The path to the window to field.

Returns

PyObject - A reference to the window, if it was open.

Scope

Client

Examples

Example 1:

This example will get the window named 'Overview' and then close it.

© 2011 Inductive Automation

```
try:
   window = system.gui.getWindow('Overview')
   system.gui.closeWindow(window)

except ValueError:
   system.gui.warningBox("The Overview window isn't open")
```

Example 2:

This example will set a value on a label component in the 'Header' window.

```
try:
   window = system.gui.getWindow('Header')
   window.getRootContainer().getComponent('Label').text = "Machine 1 Starting"

except ValueError:
   system.gui.warningBox("The Header window isn't open")
```

9.6.12 system.gui.getWindowNames

Description

Returns a list of the paths of all windows in the current project, sorted alphabetically.

Syntax

```
system.gui.getWindowNames()
```

Parameters

none

Returns

PyTuple - A tuple of strings, representing the path of each window defined in the current project.

Scope

Client

Examples

This example would open windows that begin with "Motor" and pass in the currently selected motor number.

```
motor = event.source.parent.number
windows = system.gui.getWindowNames()
for path in windows:
   if name[:5] == "Motor":
      system.gui.openWindow(path, {"motorNumber":motor})
```

9.6.13 system.gui.inputBox

Description

Opens up a popup input dialog box. This dialog box will show a prompt message, and allow the user to type in a string. When the user is done, they can press "OK" or "Cancel". If OK is pressed, this function will return with the value that they typed in. If Cancel is pressed, this function will return the value `None`.

Syntax

© 2011 Inductive Automation

system.gui.**inputBox(**message, defaultText**)**

Parameters

String `message` - The message to display for the input box.
String `defaultText` - The default text to initialize the input box with.

Returns

String - The string value that was entered in the input box.

Scope

Client

Examples

This could go in the `mouseClicked` event of a label to allow the user to change the label's text.

```
txt = system.gui.inputBox("Enter text:", event.source.text)
if txt != None:
   event.source.text = txt
```

9.6.14 system.gui.isTouchscreenModeEnabled

Description

Checks whether or not the running client's touchscreen mode is currently enabled.

Syntax

system.gui.**isTouchscreenModeEnabled()**

Parameters

none

Returns

boolean - True(1) if the client currently has touhcscreen mode activated.

Scope

Client

Examples

This example should be used in the Client Startup Script to check if this client is being run on a touch screen computer (judged by an IP address) and set touchscreen mode.

```
ipAddress = system.net.getIpAddress()
isTouchscreen = system.db.runScalarQuery("SELECT COUNT(*) FROM touchscreen_computer_ips WHERE i
if isTouchscreen and not system.gui.isTouchscreenModeEnabled():
   system.gui.setTouchscreenModeEnabled(1)
```

See also:
system.gui.setTouchscreenModeEnabled

9.6.15 system.gui.messageBox

Description

Displays an informational-style message popup box to the user.

Syntax

© 2011 Inductive Automation

system.gui.**messageBox(**message **[**, title**])**

Parameters

String `message` - The message to display.

String `title` - A title for the message box. [optional]

Returns

nothing

Scope

Client

Examples

This example will show how many hours a motor has been running when it is clicked.

```
# get the motor number
motorNumber = event.source.getPropertyValue('MotorNumber')
# retrieve the hours running from the database
hours = system.db.runScalarQuery("SELECT HoursRunning FROM MotorStatus WHERE motor=%d" % motorN

system.gui.messageBox("The motor has been running for %d hours" % motorNumber)
```

See also:
system.gui.warningBox
system.gui.errorBox

9.6.16 system.gui.moveComponent

Description

Alter's a components position to a new pair of coordinates, (x,y), a point relative to the upper-left corner of the component's parent. Note that when using relative layout, these coordinates are evaluated as if the component's size was the same size as the last time the component was saved in the Designer. This effectively means that your argument coordinates will automatically scale with relative layout.

Syntax

system.gui.**moveComponent(**component, x, y**)**

Parameters

JComponent `component` - The component to move.

int `x` - The x-coordinate to move to, relative to the upper-left corner of the component's parent container.

int `y` - The y-coordinate to move to, relative to the upper-left corner of the component's parent container.

Returns

nothing

Scope

Client

Examples

This code would go in a Timer's `propertyChange` script for animation.

```
if event.propertyName == "value":
```

```
   newX = event.newValue;
   rect = event.source.parent.getComponent("Rectangle")
   system.gui.moveComponent(rect, newX, 250)
```

See also:
system.gui.reshapeComponent
system.gui.resizeComponent

9.6.17 system.gui.passwordBox

Description

Pops up a special input box that uses a password field, so the text isn't echoed back in clear-text to the user. Returns the text they entered, or `None` if they canceled the dialog box.

Syntax

system.gui.**passwordBox(**message **[**, title**] [**, echoChar**])**

Parameters

String `message` - The message for the password prompt.
String `title` - A title for the password prompt. [optional]
String `echoChar` - A custom echo character. Defaults to: * [optional]

Returns

String - The password that was entered, or None if the prompt was canceled.

Scope

Client

Examples

This example would prompt a user for a password before opening the 'Admin' Screen.

```
password = system.gui.passwordBox("Please enter the password.")
if password == "open sesame":
   system.nav.openWindow("Admin")
```

9.6.18 system.gui.reshapeComponent

Description

Sets a component's position and size at runtime. The coordinates work in the same way as the `system.gui.moveComponent` function.

Syntax

system.gui.**reshapeComponent(**component, x, y, width, height**)**

Parameters

JComponent `component` - The component to move and resize
int `x` - The x-coordinate to move to, relative to the upper-left corner of the component's parent container.
int `y` - The y-coordinate to move to, relative to the upper-left corner of the component's parent container.
int `width` - The new width for the component
int `height` - The new height for the component

© 2011 Inductive Automation

Returns
nothing

Scope
Client

Examples

This code would go in a Timer's `propertyChange` script for animation.

```
if event.propertyName == "value":
   newX = event.newValue;
   newWidth = int(event.newValue*1.5)
   rect = event.source.parent.getComponent("Rectangle")
   system.gui.reshapeComponent(rect, newX, 150, newWidth, 80)
```

See also:
system.gui.resizeComponent
system.gui.moveComponent

9.6.19 system.gui.resizeComponent

Description

Sets a component's size at runtime. The coordinates work in the same way as the `system.gui.moveComponent` function.

Syntax

```
system.gui.resizeComponent(component, width, height)
```

Parameters
JComponent `component` - The component to resize
int `width` - The new width for the component
int `height` - The new height for the component

Returns
nothing

Scope
Client

Examples

This code would go in a Timer's propertyChange script for animation \

```
if event.propertyName == "value":
   newWidth = event.newValue;
   rect = event.source.parent.getComponent("Rectangle")
   system.gui.resizeComponent(newWidth, 80)
```

See also:
system.gui.reshapeComponent
system.gui.moveComponent

9.6.20 system.gui.setTouchscreenModeEnabled

Description

Alters a running client's touchscreen mode on the fly.

© 2011 Inductive Automation

Syntax

```
system.gui.setTouchscreenModeEnabled(enabled)
```

Parameters

boolean `enabled` - The new value for touchscreen mode being enabled.

Returns

nothing

Scope

Client

Examples

This example could be used on an input heavy window's internalFrameActivated event to remove touch screen mode.

```
if system.gui.isTouchscreenModeEnabled():
   system.gui.setTouchscreenModeEnabled(0)
```

See also:
system.gui.isTouchscreenModeEnabled

9.6.21 system.gui.showNumericKeypad

Description

Displays a modal on-screen numeric keypad, allowing for arbitrary numeric entry using the mouse, or a finger on a touchscreen monitor. Returns the number that the user entered.

Syntax

```
system.gui.showNumericKeypad(initialValue [, fontSize])
```

Parameters

Number `initialValue` - The value to start the on-screen keypad with.

int `fontSize` - The font size to display in the keypad. [optional]

Returns

Number - The value that was entered in the keypad.

Scope

Client

Examples

This function is a holdover for backwards compatibility. Input components now know when the client is in touchscreen mode and respond accordingly. This script would go in the MouseClicked or MousePressed action of a Text Field or Numeric Text Field.

© 2011 Inductive Automation

```
# For Integer Numeric Text Field:
if system.gui.isTouchscreenModeEnabled():
   event.source.intValue = system.gui.showNumericKeypad(event.source.intValue)

# For Double Numeric Text Field:
if system.gui.isTouchscreenModeEnabled():
   event.source.doubleValue = system.gui.showNumericKeypad(event.source.doubleValue)

# For Text Field:
# notice the str() and int() functions used to convert the text to a number and vice versa
# str() and int() are built-in Jython functions
if system.gui.isTouchscreenModeEnabled():
   event.source.text = str(system.gui.showNumericKeypad(int(event.source.text)))
```

9.6.22 system.gui.showTouchscreenKeyboard

Description

Displays a modal on-screen keyboard, allowing for arbitrary text entry using the mouse, or a finger on a touchscreen monitor. Returns the text that the user "typed".

Syntax

system.gui.**showTouchscreenKeyboard(**initialText **[**, fontSize**] [**, passwordMode**])**

Parameters

String `initialText` - The text to start the on-screen keyboard with.
int `fontSize` - The font size to display in the keyboard. [optional]
boolean `passwordMode` - True (1) to activate passwordmode, where the text entered isn't echoed back clear-text. [optional]

Returns

String - The text that was "typed" in the on-screen keyboard.

Scope

Client

Examples

This function is a holdover for backwards compatibility. Input components now know when the client is in touchscreen mode and respond accordingly. This would go in the MouseClicked or MousePressed action of a Text Field or similar component.

```
if system.gui.isTouchscreenModeEnabled():
   event.source.text = system.gui.showTouchscreenKeyboard(event.source.text)
```

9.6.23 system.gui.warningBox

Description

Displays a message to the user in a warning style pop-up dialog.

Syntax

system.gui.**warningBox(**message **[**, title**])**

Parameters

String `message` - The message to display in the warning box.

© 2011 Inductive Automation

String `title` - The title for the warning box. [optional]

Returns
nothing

Scope
Client

Examples

This code show a yellow popup box similar to the system.gui.messageBox function.

```
# Start the motor, or, warn the user if in wrong mode
runMode = event.source.parent.getPropertyValue('RunMode')
if runMode == 1: Cannot start the motor in mode #1
   system.gui.warningBox("Cannot start the motor, current mode is <B>VIEW MODE</B>")
else:
   system.db.runUpdateQuery("UPDATE MotorControl SET MotorRun=1")
```

See also:
system.gui.messageBox
system.gui.errorBox

9.7 system.nav

9.7.1 system.nav.centerWindow

Description

Given a window path, or a reference to a window itself, it will center the window. The window should be floating an non-maximized. If the window can't be found, this function will do nothing.

Syntax

```
system.nav.centerWindow(window)
```

Parameters
FPMIWindow `window` - A reference to the window to center.

Returns
nothing

Scope
Client

```
system.nav.centerWindow(windowPath)
```

Parameters
String `windowPath` - The path of the window to center.

Returns
nothing

Scope
Client

Examples

```
#This example centers the window named 'Overview'.
system.nav.centerWindow('Overview')
```

See also:

© 2011 Inductive Automation

system.nav.openWindow

9.7.2 system.nav.closeParentWindow

Description

Closes the parent window given a component event object.

Syntax

```
system.nav.closeParentWindow(event)
```

Parameters

EventObject `event` - A component event object. The enclosing window for the component will be closed.

Returns

nothing

Scope

Client

Examples

```
#This code would be placed in the actionPerformed event of a button, and would close the window
system.nav.closeParentWindow(event)
```

9.7.3 system.nav.closeWindow

Description

Given a window path, or a reference to a window itself, it will close the window. If the window can't be found, this function will do nothing.

Syntax

```
system.nav.closeWindow(windowPath)
```

Parameters

String `windowPath` - The path of a window to close.

Returns

nothing

Scope

Client

```
system.nav.closeWindow(window)
```

Parameters

FPMIWindow `window` - A reference to the window to close.

Returns

nothing

Scope

Client

Examples

Example 1:

This example would get the window named 'Overview' and then close it.

© 2011 Inductive Automation

```
# If the window isn't open, show a warning
try:
   window = system.gui.getWindow('Overview')
   system.nav.closeWindow(window)
except ValueError:
   system.gui.warningBox("The Overview window isn't open")
```

Example 2:

This example would close the window named 'Overview' in one step.

```
# If the window isn't open, the call to closeWindow will have no effect
system.nav.closeWindow('Overview')
```

9.7.4 system.nav.getCurrentWindow

Description

Returns the path of the current "main screen" window, which is defined as the maximized window. With the Typical Navigation Strategy, there is only ever one maximized window at a time.

Syntax

```
system.nav.getCurrentWindow()
```

Parameters

none

Returns

String - The path of the current "main screen" window - the maximized window.

Scope

Client

Examples

```
# This code could run in a global timer script.
# After a 5-minute timeout, navigate back to the home screen
if system.util.getInactivitySeconds() > 300 and system.nav.getCurrentWindow() != "HomeScreen":
   system.nav.swapTo("HomeScreen")
```

9.7.5 system.nav.goBack

Description

When using the Typical Navigation Strategy, this function will navigate back to the previous main screen window.

Syntax

```
system.nav.goBack()
```

Parameters

none

Returns

PyObject - The window that was returned to

Scope

© 2011 Inductive Automation

Client

Examples

This code would go in a button to move to the previous screen.

```
system.nav.goBack()
```

9.7.6 system.nav.goForward

Description

When using the Typical Navigation Strategy, this function will navigate "forward" to the last main-screen window the user was on when they executed a `system.nav.goBack()`.

Syntax

```
system.nav.goForward()
```

Parameters

none

Returns

PyObject - The window that was returned to

Scope

Client

Examples

This code would go in a button to move to the last screen that used system.nav.goBack().

```
system.nav.goForward()
```

9.7.7 system.nav.goHome

Description

When using the Typical Navigation Strategy, this function will navigate to the "home" window. This is automatically detected as the first main-screen window shown in a project.

Syntax

```
system.nav.goHome()
```

Parameters

none

Returns

PyObject - A reference to the home window that was navigated to.

Scope

Client

Examples

This code would go in a button to move to the Home screen.

```
system.nav.goHome()
```

© 2011 Inductive Automation

9.7.8 system.nav.openWindow

Description

Opens the window with the given path. If the window is already open, brings it to the front. The optional `params` dictionary contains key:value pairs which will be used to set the target window's root container's dynamic variables.

For instance, if the window that you are opening is named "TankDisplay" has a dynamic variable in its root container named "TankNumber", then calling `system.nav.openWindow ("TankDisplay", {"TankNumber" : 4})` will open the "TankDisplay" window and set `Root Container.TankNumber` to four. This is useful for making parameterized windows, that is, windows that are re-used to display information about like pieces of equipment. See also: Parameterized Windows.

Syntax

system.nav.**openWindow(**path **[**, params**])**

Parameters

String `path` - The path to the window to open.

PyDictionary `params` - A dictionary of parameters to pass into the window. The keys in the dictionary must match dynamic property names on the target window's root container. The values for each key will be used to set those properties. [optional]

Returns

PyObject - A reference to the opened window.

Scope

Client

Examples

Example 1:

```
# This is the simplest form of openWindow
system.nav.openWindow("SomeWindowName")
```

Example 2:

```
# A more complex example - a setpoint screen for multiple valves that opens centered
titleText = "Third Valve Setpoints"
tankNo = system.nav.openWindow("ValveSetPts", {"valveNum":3, "titleText":titleText})
system.nav.centerWindow("ValveSetPts")
```

9.7.9 system.nav.openWindowInstance

Description

Operates exactly like system.nav.openWindow, except that if the named window is already open, then an *additional instance* of the window will be opened. There is no limit to the number of additional instances of a window that you can open.

Syntax

system.nav.**openWindowInstance(**path **[**, params**])**

© 2011 Inductive Automation

Parameters

String `path` - The path to the window to open.

PyDictionary `params` - A dictionary of parameters to pass into the window. The keys in the dictionary must match dynamic property names on the target window's root container. The values for each key will be used to set those properties. [optional]

Returns

PyObject - A reference to the opened window.

Scope

Client

Examples

This example would open three copies of a single HOA popup screen.

```
system.nav.openWindowInstance("HOA" {machineNum:3})
system.nav.openWindowInstance("HOA" {machineNum:4})
system.nav.openWindowInstance("HOA" {machineNum:5})
```

9.7.10 system.nav.swapTo

Description

Performs a window swap from the current *main screen* window to the window specified. Swapping means that the opened window will take the place of the closing window - in this case it will be maximized. See also: Typical Navigation Strategy.

Syntax

```
system.nav.swapTo(path [, params])
```

Parameters

String `path` - The path of a window to swap to.

PyDictionary `params` - A dictionary of parameters to pass into the window. The keys in the dictionary must match dynamic property names on the target window's root container. The values for each key will be used to set those properties. [optional]

Returns

PyObject - A reference to the swapped-to window.

Scope

Client

Examples

Example 1:

This code would go in a button's ActionPerformed event to swap out of the current window and into a window named MyWindow

```
system.nav.swapTo("MyWindow")
```

Example 2:

This code would go in a button's ActionPerformed event to swap out of the current window and into a window named MyWindow. It also looks at the selected value in a dropdown menu and passes that value into the new window.

© 2011 Inductive Automation

```
# MyWindow's Root Container must have a dynamic property named "paramValue"
dropdown = event.source.parent.getComponent("Dropdown")
system.nav.swapTo("MyWindow", {"paramValue":dropdown.selectedValue)
```

See also:
system.nav.swapWindow

9.7.11 system.nav.swapWindow

Description

Performs a window *swap*. This means that one window is closed, and another is opened and takes its place - assuming its size, floating state, and maximization state. This gives a seamless transition - one window seems to simply turn into another.

Syntax

system.nav.**swapWindow(**swapFromPath, swapToPath **[**, params**])**

Parameters

String `swapFromPath` - The path of the window to swap from. Must be a currently open window, or this will act like an openWindow.

String `swapToPath` - The name of the window to swap to.

PyDictionary `params` - A dictionary of parameters to pass into the window. The keys in the dictionary must match dynamic property names on the target window's root container. The values for each key will be used to set those properties. [optional]

Returns

PyObject - A reference to the swapped-to window.

Scope

Client

system.nav.**swapWindow(**event, swapToPath **[**, params**])**

Parameters

EventObject `event` - A component event whose enclosing window will be used as the "swap-from" window.

String `swapToPath` - The name of the window to swap to.

PyDictionary `params` - A dictionary of parameters to pass into the window. The keys in the dictionary must match dynamic property names on the target window's root container. The values for each key will be used to set those properties. [optional]

Returns

PyObject - A reference to the swapped-to window.

Scope

Client

Examples

This function works like system.nav.swapTo except that you can specify the source and destination for the swap.

Example 1:

```
# This code would go in a button's ActionPerformed event to swap out of the
# window containing the button and into a window named MyWindow
system.nav.swapWindow(event, "MyWindow")
```

© 2011 Inductive Automation

Example 2:

```
# This code would swap from window named WindowA to a window named WindowB
system.nav.swapWindow("WindowA", "WindowB")
```

Example 3:

```
# This code would swap from window named WindowA to a window named WindowB.
# It also looks at the two calendar popup controls and passes the two selected dates to
# WindowB. WindowB's Root Container must have dynamic properties named "startDate" and
# "endDate"
date1 = event.source.parent.getComponent("Start Date").date
date2 = event.source.parent.getComponent("End Date").date
system.nav.swapWindow("WindowA", "WindowB", {"startDate":date1, "endDate":date2})
```

See also:
system.nav.swapTo

9.8 system.net

9.8.1 system.net.getExternalIpAddress

Description

Returns the client's IP address, *as it is detected by the Gateway*. This means that this call will communicate with the Gateway, and the Gateway will tell the clienth what IP address its incoming traffic is coming from. If you have a client behind a NAT router, then this address will be the WAN address of the router instead of the LAN address of the client, which is what you'd get with `system.net.getIpAddress`.

Syntax

```
system.net.getExternalIpAddress()
```

Parameters
none

Returns
String - A text representation of the client's IP address, as detected by the Gateway

Scope
Client

Examples

Put this script on a navigation button to restrict users from opening a specific page.

```
ip = sytem.net.getExternalIpAddress()
#check if this matches the CEO's IP address
if ip == "66.102.7.104":
   system.nav.swapTo("CEO Dashboard")
else:
   system.nav.swapTo("Manager Dashboard")
```

See also:
system.net.getHostName
system.net.getIpAddress

© 2011 Inductive Automation

9.8.2 system.net.getHostName

Description

Returns the host name of the computer that the client is currently running on. On Windows, this is typically the "computer name". For example, might return `EAST_WING_WORKSTATION` or `bobs-laptop`.

Syntax

```
system.net.getHostName()
```

Parameters

none

Returns

String - The hostname of the local machine. This is the computer that the script is being executed on - may be a Client or the Gateway depending on the script context.

Scope

All

Examples

Put this script on a navigation button to link dedicated machines to specific screens.

```
comp = sytem.net.getHostName()
#check which line this client is tied to
if comp == "Line1Computer":
   system.nav.swapTo("Line Detail", {"line":1})
elif comp == "Line2Computer":
   system.nav.swapTo("Line Detail", {"line":2})
else:
   system.nav.swapTo("Line Overview")
```

See also:
system.net.getExternalIpAddress
system.net.getIpAddress

9.8.3 system.net.getIpAddress

Description

Returns the IP address of the computer the client is running on, as it appears to the client. See also: `system.net.getExternalIpAddress()`.

Syntax

```
system.net.getIpAddress()
```

Parameters

none

Returns

String - Returns the IP address of the local machine, as it sees it.

Scope

All

Examples

© 2011 Inductive Automation

Put this script on a navigation button to link dedicated machines to specific screens.

```
ip = sytem.net.getIpAddress()
#check which line this client is tied to
if ip == "10.1.10.5":
   system.nav.swapTo("Line Detail", {"line":1})
elif ip == "10.1.10.6":
   system.nav.swapTo("Line Detail", {"line":2})
else:
   system.nav.swapTo("Line Overview")
```

See also:
system.net.getExternalIpAddress
system.net.getHostName

9.8.4 system.net.httpGet

Description

Retrieves the document at the given URL using the HTTP GET protocol. The document is returned as a string. For example, if you use the URL of a website, you'll get the same thing you'd get by going to that website in a browser and using the browser's "View Source" function.

Syntax

```
system.net.httpGet(url)
```

Parameters
String `url` - The URL to retrieve.

Returns
String - The content found at the given URL.

Scope
All

Examples

Example 1:

```
# This code would return the source for Google's homepage
source = system.net.httpGet("http://www.google.com")
print source
```

Example 2:

```
# This code would query Yahoo Weather for the temperature at
# Inductive Automation's headquarters in Sacramento, CA
# and then find the current temperature using a regular expression

response = system.net.httpGet("http://xml.weather.yahoo.com/forecastrss?p=95818")

# import Python's regular expression library
import re

# NOTE - if you've never seen regular expressions before, don't worry, they look
# confusing even to people who use them frequently.
pattern = re.compile('.*?<yweather:condition (.*?)/>', re.DOTALL)
match = pattern.match(response)
```

© 2011 Inductive Automation

```
if match:
   subText = match.group(1)
   condition = re.compile('.*?text="(.*?)"').match(subText).group(1)
   temp = re.compile('.*?temp="(.*?)"').match(subText).group(1)
   print "Condition: ", condition
   print "Temperature (F): ", temp
else:
   print 'Weather service format changed'
```

9.8.5 system.net.httpPost

Description

Retrieves the document at the given URL using the HTTP POST protocol. If a parameter dictionary argument is specified, the entries in the dictionary will encoded in "application/x-www-form-urlencoded" format, and then posted. You can post arbitrary data as well, but you'll need to specify the MIME type. The document is then returned as a string.

Syntax

system.net.**httpPost(**url, postParams**)**

Parameters

String `url` - The URL to post to.
PyDictionary `postParams` - A dictionary of name: value key pairs to use as the post data.

Returns

String - The content returned for the POST operation.

Scope

All

system.net.**httpPost(**url, contentType, postData**)**

Parameters

String `url` - The URL to post to.
String `contentType` - The MIME type to use in the HTTP "Content-type" header.
String `postData` - The raw data to post via HTTP.

Returns

String - The content returned for the POST operation.

Scope

All

Examples

Example 1:

```
# This code posts a name (first and last) to the post testing page at
# "http://www.snee.com/xml/crud/posttest.cgi", and returns the resulting page as a string.
page = system.net.httpPost("http://www.snee.com/xml/crud/posttest.cgi", {"fname":"Billy", "lnam
print page
```

Example 2:

```
# This code sends an XML message to a hypothetical URL.
message = "<MyMessage><MyElement>here is the element</MyElement></MyMessage>"
system.net.httpPost("http://www.posttome.xyz/posthere", "text/xml", message)
```

© 2011 Inductive Automation

9.8.6 system.net.openURL

Description

Opens the given URL outside of the currently running Client in whatever application the host operating system deems appropriate. For example, the URL:

```
"http://www.inductiveautomation.com"
```

... will open in the default web browser, whereas this one:

```
"file://C:\Report.pdf"
```

... will likely open in Adobe Acrobat. The Windows network-share style path like:

```
"\\Fileserver\resources\machine_manual.pdf"
```

... will work as well (in Windows).

Be careful not to use this function in a full-screen client, as launching an external program will break your full-screen exclusive mode.

Syntax

```
system.net.openURL(url [, useApplet])
```

Parameters

String `url` - The URL to open in a web browser.

boolean `useApplet` - If set to true (1), and the client is running as an Applet, then the browser instance that launched the applet will be used to open the URL. [optional]

Returns

nothing

Scope

Client

Examples

Example 1:

```
# This code would open a web page
system.net.openURL("http://www.google.com")
```

Example 2:

```
# This code would open a PDF document from a Windows-based file server
# Note the double backslashes are needed because backslash is the escape character for Jython
system.net.openURL("\\\\MyServer\\MyDocs\\document.pdf")
```

9.8.7 system.net.sendEmail

Description

Sends an email through the given SMTP server. Note that this email is relayed first through the Gateway - the client host machine doesn't need network access to the SMTP server.

You can send text messages to cell phones and pagers using email. Contact your cell carrier for details. If you had a Verizon cell phone with phone number (123) 555-8383, for example, your text messaging email address would be: 1235558383@vtext.com. Try it out!

© 2011 Inductive Automation

This function accepts *keyword-style* invocation. See also: Functions / Keyword Invocation

Syntax

system.net.**sendEmail(**smtp, from, subject, body, html, to, attachmentNames, attachmentData, timeout, username, password**)**

Parameters

String `smtp` - The address of an SMTP server to send the email through, like "mail.example.com"
String `from` - An email address to have the email come from.
String `subject` - The subject line for the email
String `body` - The body text of the email.
Boolean `html` - A flag indicating whether or not to send the email as an HTML email. Will auto-detect if omitted.
String[] `to` - A list of email addresses to send to.
String[] `attachmentNames` - A list of attachment names.
byte[][] `attachmentData` - A list of attachment data, in binary format.
Integer `timeout` - A timeout for the email, specified in milliseconds. Defaults to 5 minutes (60,000*5)
String `username` - If specified, will be used to authenticate with the SMTP host.
String `password` - If specified, will be used to authenticate with the SMTP host.

Returns

nothing

Scope

All

Examples

Example 1:

```
# This code would send a simple plain-text email to a single recipient, with no attachments
body = "Hello, this is an email."
recipients = ["bobsmith@mycompany.com"]
system.net.sendEmail("mail.mycompany.com", "myemail@mycompany.com", "Here is the email!", body,
```

Example 2:

```
# This code would send an HTML-formatted email to multiple recipients (including cellphones) wi
body = "<HTML><BODY><H1>This is a big header</H1>And this text is <font color='red'>red</font><
recipients = ["bobsmith@mycompany.com", "1235558383@vtext.com", "sally@acme.org", "1235557272@v
myuser = "mycompany"
mypass = "1234"
system.net.sendEmail(smtp="mail.mycompany.com", from="myemail@mycompany.com", subject="Here is
```

Example 3:

```
# This code ask the user for an attachment file and attach the file.
filePath = fpmi.file.openFile()
if filePath != None:
   fileName = filePath.split("\\")[-1] # This gets the filename without the C:\folder stuff
   fileData = fpmi.file.readFileAsBytes(filePath)
   smtp = "mail.mycompany.com"
   sender = "myemail@mycompany.com"
   subject = "Here is the file you requested"
   body = "Hello, this is an email."
```

© 2011 Inductive Automation

```
recipients = ["bobsmith@mycompany.com"]
system.net.sendEmail(smtp, sender, subject, body, 0, recipients, [fileName], [fileData])
```

9.9 system.opc

9.9.1 system.opc.getServerState

Description

Retreives the current state of the given OPC server connection. If the given server is not found, the return value will be `None`. Otherwise, the return value will be one of these strings:

UNKNOWN	FAULTED
CONNECTING	CLOSED
CONNECTED	DISABLED

Syntax

```
system.opc.getServerState(opcServer)
```

Parameters

String `opcServer` - The name of an OPC server connection.

Returns

String - A string representing the current state of the connection, or None if the connection doesn't exist.

Scope

All

9.9.2 system.opc.readValue

Description

Reads a single value directly from an OPC server connection. The address is specified as a string, for example, `[MyDevice]N11/N11:0`
The object returned from this function has three attributes: `value`, `quality`, and `timestamp`. The `value` attribute represents the current value for the address specified. The `quality` attribute is an OPC-UA status code. You can easily check a good quality vs a bad quality by calling the `isGood()` function on the quality object. The `timestamp` attribute is Date object that represents the time that the value was retrieved at.

Syntax

```
system.opc.readValue(opcServer, itemPath)
```

Parameters

String `opcServer` - The name of the OPC server connection in which the item resides.

String `itemPath` - The item path, or address, to read from.

Returns

QualifiedValue - An object that contains the value, quality, and timestamp returned from the OPC server for the address specified.

Scope

All

© 2011 Inductive Automation

9.9.3 system.opc.readValues

Description

This function is equivalent to the `system.opc.readValue` function, except that it can operate in bulk. You can specify a list of multiple addresses to read from, and you will receive a list of the same length, where each entry is the qualified value object for the corresponding address.

Syntax

```
system.opc.readValues(opcServer, itemPaths)
```

Parameters

String `opcServer` - The name of the OPC server connection in which the items reside.
String[] `itemPaths` - A list of strings, each representing an item path, or address to read from.

Returns

QualifiedValue[] - A sequence of objects, one for each address specified, in order. Each object will contains the value, quality, and timestamp returned from the OPC server for the corresponding address.

Scope

All

9.9.4 system.opc.writeValue

Description

Writes a value directly through an OPC server connection. Will return an OPC-UA status code object. You can quickly check if the write succeeded by calling `isGood()` on the return value from this function.

Syntax

```
system.opc.writeValue(opcServer, itemPath, value)
```

Parameters

String `opcServer` - The name of the OPC server connection in which the item resides.
String `itemPath` - The item path, or address, to write to.
Object `value` - The value to write to the OPC item.

Returns

StatusCode - The status of the write. Use returnValue.isGood() to check if the write succeeded.

Scope

All

9.9.5 system.opc.writeValues

Description

This function is a bulk version of `system.opc.writeValue`. It takes a list of addresses and a list of objects, which must be the same length. It will write the corresponding object to the corresponding address in bulk. It will return a list of status codes representing the individual write success or failure for each corresponding address.

Syntax

© 2011 Inductive Automation

```
system.opc.writeValues(opcServer, itemPaths, values)
```

Parameters

String `opcServer` - The name of the OPC server connection in which the items reside.
String[] `itemPaths` - A list of item paths, or addresses, to write to.
Object[] `values` - A list of values to write to each address specified.

Returns

StatusCode[] - An array of status codes, each entry corresponding in order to the addresses specified.

Scope

All

9.10 system.print

9.10.1 system.print.createImage

Description

Advanced Function. Takes a snapshot of a component and creates a Java BufferedImage out of it. You can use javax.imageio.ImageIO to turn this into bytes that can be saved to a file or a BLOB field in a database.

Syntax

```
system.print.createImage(component)
```

Parameters

Component `component` - The component to render.

Returns

BufferedImage - A java.awt.image.BufferedImage representing the component.

Scope

Client

9.10.2 system.print.createPrintJob

Description

Provides a general printing facility for printing the contents of a window or component to a printer. The general workflow for this function is that you create the print job, set the options you'd like on it, and then call `print()` on the job.

For printing reports or tables, use those components' dedicated print() functions.

The PrintJob object that this function returns has the following properties that can be set:

showPrintDialog	If true (1), then the print dialog window will be shown before printing. This allows users to specify printing options like orientation, printer, paper size, margins, etc. [default: 1]
fitToPage	If the component is too wide or tall to fit on a page, it will be proportionately zoomed out until it fits into the page. [default: 1]
zoomFactor	If greater than zero, this zoom factor will be used to zoom the printed image in or out. For example, if this is 0.5, the printed image will be half size. If used, this zoom factor overrides the fitToPage parameter.

© 2011 Inductive Automation

	[default: -1.0]
orientation	Either `system.print.PORTRAIT` or `system.print.LANDSCAPE` [default: `system.print.PORTRAIT`]
pageWidth	The width of the paper in inches. [default: 8.5]
pageHeight	The height of the paper in inches. [default: 11]
leftMargin, rightMargin, topMargin, bottomMargin	The margins, specified in inches. [default: 0.75]

You can set all of the margins at once with `job.setMargins(number)`, and you initiate the printing with `job.print()`.

Syntax

```
system.print.createPrintJob(component)
```

Parameters

Component `component` - The component that you'd like to print.

Returns

JythonPrintJob - A print job that can then be customized and started.

Scope

Client

Examples

Put this code on a button to print out an image of the container the button is in

```
job = system.print.createPrintJob(event.source.parent)
job.setMargins(0.5)
job.zoomFactor = 0.75
job.showPageFormat = 0
job.print()
```

9.10.3 system.print.printToImage

Description

This function prints the given component (such as a graph, container, entire window, etc) to an image file, and prompts the user to save the file to their hard drive.

Syntax

```
system.print.printToImage(component [, filename])
```

Parameters

Component `component` - The component to render.

String `filename` - A filename to save the image as. [optional]

Returns

nothing

Scope

Client

Examples

This code would go on a button and *save* an image of the container that it is in.

```
system.print.printToImage(event.source.parent, "Screen.jpg")
```

© 2011 Inductive Automation

9.11 system.security

9.11.1 system.security.getRoles

Description

Finds the roles that the currently logged in user has, returns them as a Python tuple of strings.

Syntax

```
system.security.getRoles()
```

Parameters

none

Returns

PyTuple - A list of the roles (strings) that are assigned to the current user.

Scope

Client

Examples

This would run on a button to prevent certain users from opening a window

```
if "Supervisor" in system.security.getRoles():
   system.nav.openWindow("ManagementOnly")
else:
   system.gui.errorBox("You don't have sufficient privileges to continue")
```

9.11.2 system.security.getUsername

Description

Returns the currently logged-in username.

Syntax

```
system.security.getUsername()
```

Parameters

none

Returns

String - The current user.

Scope

Client

Examples

This code would run on a startup script and do special logic based upon who was logging in

```
name = system.security.getUsername()
if name == 'Bob':
   system.nav.openWindow("BobsHomepage")
else:
   system.nav.openWindow("NormalHomepage")
```

© 2011 Inductive Automation

9.11.3 system.security.isScreenLocked

Description

Returns whether or not the screen is currently locked.

Syntax

system.security.**isScreenLocked()**

Parameters

none

Returns

boolean - A flag indicating whether or not the screen is currently locked.

Scope

Client

Examples

This would run in a timer script to lock the screen after 15 seconds of inactivity, and then log the user out after 30 seconds of inactivity.

```
if system.util.getInactivitySeconds() > 15 and not system.security.isScreenLocked():
   system.security.lockScreen()
elif system.util.getInactivitySeconds() > 30:
   system.security.logout()
```

9.11.4 system.security.lockScreen

Description

Used to put a running client in lock-screen mode. The screen can be unlocked by the user with the proper credentials, or by scripting via the `system.security.unlockScreen()` function.

Syntax

system.security.**lockScreen(**[obscure]**)**

Parameters

boolean `obscure` - If true(1), the locked screen will be opaque, otherwise it will be partially visible. [optional]

Returns

nothing

Scope

Client

Examples

This would run in a timer script to lock the screen after 15 seconds of inactivity, and then log the user out after 30 seconds of inactivity.

```
if system.util.getInactivitySeconds() > 15 and not system.security.isScreenLocked():
   system.security.lockScreen()
elif system.util.getInactivitySeconds() > 30:
   system.security.logout()
```

© 2011 Inductive Automation

9.11.5 system.security.logout

Description

Shuts-down the currently running client and brings the client to the login screen.

Syntax

```
system.security.logout()
```

Parameters

none

Returns

nothing

Scope

Client

Examples

This would run in a timer script to log the user out after 30 seconds of inactivity.

```
if system.util.getInactivitySeconds() > 30:
   system.security.logout()
```

See also:
system.util.getInactivitySeconds

9.11.6 system.security.switchUser

Description

Attempts to switch the current user on the fly. If the given username and password fail, this function will return false. If it succeeds, then all currently opened windows are closed, the user is switched, and windows are then re-opened in the states that they were in.

If an event object is passed to this function, the parent window of the event object will *not* be re-opened after a successful user switch. This is to support the common case of having a switch-user screen that you want to disappear after the switch takes place.

Syntax

```
system.security.switchUser(username, password, event, hideError)
```

Parameters

String `username` - The username to try and switch to.
String `password` - The password to authenticate with.
EventObject `event` - If specified, the enclosing window for this event's component will be closed in the switch user process.
Boolean `hideError` - If true (1), no error will be shown if the switch user function fails. (default: 0)

Returns

boolean - false(0) if the switch user operation failed, true (1) otherwise.

Scope

Client

© 2011 Inductive Automation

Examples

This script would go on a button in a popup window used to switch users without logging out of the client.

```
# Pull the username and password from the input components
uname = event.source.parent.getComponent("Username").text
pwd = event.source.parent.getComponent("Password").text

# Call switchUser. The event object is passed to this
# function so that if the username and password work,
# this window will be closed before the switch occurs.
success= system.security.switchUser(uname,pwd,event)

# If the login didn't work, give input focus back to the
# username component, so that the user can try again
if not success:
   event.source.parent.getComponent("Username").requestFocusInWindow()
```

9.11.7 system.security.unlockScreen

Description

Unlocks the client, if it is currently in lock-screen mode.

Syntax

```
system.security.unlockScreen()
```

Parameters

none

Returns

nothing

Scope

Client

Examples

This code would go in a global script to automatically unlock the screen on a specific computer

```
comp = system.net.getHostName()
if comp == 'Line 1':
   system.security.unlockScreen()
```

9.12 system.tag

9.12.1 system.tag.getTagValue

Description

Returns the value of the tag at the given path.

Syntax

```
system.tag.getTagValue(tagPath)
```

Parameters

String `tagPath` - The tag path to retrieve. If the property is omitted, Value is assumed.

© 2011 Inductive Automation

Returns
Object - The value for the given tag path.

Scope
All

Examples
This example would get a tag value and display it in a message box.

```
val = system.tag.getTagValue("[]EastSection/ValveG/HOA_bit")
system.gui.messageBox("The value is %d" % val)
```

9.12.2 system.tag.getTagValues

Syntax

```
system.tag.getTagValues(tagPaths)
```

Parameters
String[] `tagPaths`

Returns
Object[]

Scope
All

9.12.3 system.tag.isOverlaysEnabled

Description
Returns whether or not the current client's quality overlay system is currently enabled.

Syntax

```
system.tag.isOverlaysEnabled()
```

Parameters
none

Returns
boolean - True (1) if overlays are currently enabled.

Scope
Client

9.12.4 system.tag.queryTagHistory

Description
Issues a query to to the SQLTags Historian. Querying tag history involves specifying the tags and the date range, as well as a few optional parameters. The SQLTags historian will find the relevant history and then interpolate and aggregate it together into a coherent, tabular result set.

This function takes a list of strings, where each string is a *tag path*, like `"Tanks/Tank5"` or `"[OracleProvider]Sump/Out2"`. See also: Tag Paths.

© 2011 Inductive Automation

The return size determines how the underlying data is aggregated and/or interpolated. If a distinct return size is specified, that will be the number of rows in the resulting dataset. The special numbers 0 and -1 mean "Natural" and "On-Change", respectively. "Natural" calculates a return size based on the rate of the logging historical scan classes. For example, if you query 1 hour of data for a scan class logging every minute, the natural return size is 60. "On-Change means that you'll get an entry whenever any of the tags under consideration have changed.

The aggregation mode is used when the data is denser than what you asked for - there is more than 1 sample per time slice in the range you're requesting. "MinMax" will return two entries per time slice - the min and the max. "Average" will return the average value of all samples in that time slice.

This function accepts *keyword-style* invocation. See also: Functions / Keyword Invocation

Syntax

system.tag.**queryTagHistory(**paths, startDate, endDate, returnSize, aggregationMode, returnFormat, columnNames**)**

Parameters

PySequence `paths` - An array of tag paths (strings) to query. Each tag path specified will be a column in the result dataset.

Date `startDate` - The earliest value to retrieve. If omitted, 8 hours before current time is used.

Date `endDate` - The latest value to retrieve. If omitted, current time is used.

Integer `returnSize` - The number of samples to return. -1 will return values as they changed, and 0 will return the "natural" number of values based on the logging rates of the scan class (es) involved. -1 is the default.

String `aggregationMode` - The mode to use when aggregating multiple samples into one time slice. Must be one of "Average" or "MinMax".

String `returnFormat`

PySequence `columnNames`

Returns

Dataset - A dataset representing the historian values for the specified tag paths. The first column will be the timestamp, and each column after that represents a tag.

Scope

All

9.12.5 system.tag.setOverlaysEnabled

Description

Enables or disables the component quality overlay system.

Syntax

system.tag.**setOverlaysEnabled(**enabled**)**

Parameters

boolean `enabled` - True (1) to turn on tag overlays, false (0) to turn them off.

Returns

nothing

Scope

Client

© 2011 Inductive Automation

9.12.6 system.tag.writeToTag

Description

Writes a value to a tag. Note that this function writes *asynchronously*. This means that the function does not wait for the write to occur before returning - the write occurs sometime later on a different thread.

Syntax

system.tag.**writeToTag(**tagPath, value **[**, suppressErrors**])**

Parameters

String `tagPath` - The path of the tag to write to.
Object `value` - The value to write.
boolean `suppressErrors` - A flag indicating whether or not to supress errors. (client-only). [optional]

Returns

int - 0 if the write failed immediately, 1 if it succeeded immediately, and 2 if it is pending.

Scope

All

Examples

This code would go on a property change event for a numeric text field to calculate and write a value to a tag.

```
if event.propertyName == intValue:
    calcValue = event.newValue * 2.5
    system.tag.writeToTag("[]Tanks/tankHiSP",calcValue)
```

9.12.7 system.tag.writeToTagSynchronous

Description

Writes a value to a tag, synchronously. This means that you know at the end of this function whether or not the write succeeded or not. However, this function cannot be called from the event dispatch thread, which means that it cannot be called directly from a GUI event like a button press, without wrapping it in a system.util.invokeAsynchronous. You can call this from project event scripts like timer scripts.

Syntax

system.tag.**writeToTagSynchronous(**tagPath, value **[**, timeout**])**

Parameters

String `tagPath` - The path of the tag to write to.
Object `value` - The value to write.
int `timeout` [optional]

Returns

nothing

Scope

All

© 2011 Inductive Automation

9.12.8 system.tag.writeToTags

Syntax

```
system.tag.writeToTags(tagPaths, ??)
```

Parameters

String[] `tagPaths`
Object[] `??`

Returns

int[]

Scope

All

9.13 system.util

9.13.1 system.util.beep

Description

Tells the computer to make a "beep" sound.

Syntax

```
system.util.beep()
```

Parameters

none

Returns

nothing

Scope

All

9.13.2 system.util.execute

Description

Executes the given commands via the operating system, in a separate process The commands argument is an array of strings. The first string is the program to execute, with subsequent strings being the arguments to that command.

Syntax

```
system.util.execute(commands)
```

Parameters

String[] `commands` - A list containing the command (1st entry) and associated arguments (remaining entries) to execute.

Returns

nothing

Scope

All

© 2011 Inductive Automation

Examples

```
# This code would work on a Windows system to play a sound file.
system.util.execute(["sndrec32", "/play", "/close", "/embedding", "C:\\somethingwrong.wav"])
```

9.13.3 system.util.exit

Description

Exits the running client, as long as the *shutdown intercept* script doesn't cancel the shutdown event. Set `force` to true to not give the shutdown intercept script a chance to cancel the exit. Note that this will quit the Client completely. you can use `system.security.logout()` to return to the login screen.

Syntax

system.util.**exit(**[force]**)**

Parameters

boolean `force` - If true (1), the shutdown-intercept script will be skipped. Default is false (0). [optional]

Returns

nothing

Scope

Client

Examples

```
# This code would exit the Ignition Runtime client after confirming with the user.
if system.gui.confirm("Are you sure you want to exit?"):
    system.util.exit()
```

9.13.4 system.util.getClientId

Description

Returns a hex-string that represents a number unique to the running client's session. You are guaranteed that this number is unique between all running clients.

Syntax

system.util.**getClientId()**

Parameters

none

Returns

String - A special code representing the client's session in a unique way.

Scope

Client

Examples

```
# This code would print the current client's id to the debug console.
id = system.util.getClientId()
print id
```

© 2011 Inductive Automation

9.13.5 system.util.getConnectTimeout

Description

Returns the connect timeout in milliseconds for all client-to-gateway communication. This is the maximum amount of time that communication operations to the Gateway will be given to connect. The default is 10,000ms (10 seconds).

Syntax

```
system.util.getConnectTimeout()
```

Parameters
none

Returns
int - The current connect timeout, in milliseconds. Default is 10,000 (ten seconds)

Scope
Client

Examples

```
# This code would print out the current connect timeout
print system.util.getConnectTimeout()
```

9.13.6 system.util.getConnectionMode

Description

Retrieves this client session's current connection mode. 3 is read/write, 2 is read-only, and 1 is disconnected.

Syntax

```
system.util.getConnectionMode()
```

Parameters
none

Returns
int - The current connection mode for the client.

Scope
Client

9.13.7 system.util.getEdition

Description

Returns the "edition" of the Vision client - "standard", "limited", or "panel".

Syntax

```
system.util.getEdition()
```

Parameters
none

© 2011 Inductive Automation

Returns
String - The edition of the Vision module that is running the client.

Scope
Client

9.13.8 system.util.getGatewayAddress

Description

Returns the address of the gateway that the client is currently communicating with.

Syntax

```
system.util.getGatewayAddress()
```

Parameters
none

Returns
String - the address of the Gateway that the client is communicating with.

Scope
Client

Examples

```
# This code would open up the Ignition gateway config page.
address = system.util.getGatewayAddress()
system.net.openURL("%s/web/config/" % address)
```

9.13.9 system.util.getInactivitySeconds

Description

Returns the number of seconds since any keyboard or mouse activity. Note - this function will always return zero in the Designer.

Syntax

```
system.util.getInactivitySeconds()
```

Parameters
none

Returns
long - The number of seconds the mouse and keyboard have been inactive for this client.

Scope
Client

Examples

```
# This code could run in a global timer script.
# After a 5-minute timeout, navigate back to the home screen
if system.util.getInactivitySeconds() > 300 and system.nav.getCurrentWindow() != "HomeScreen":
   system.nav.swapTo("HomeScreen")
```

© 2011 Inductive Automation

9.13.10 system.util.getProjectName

Description

Returns the name of the project that is currently being run.

Syntax

```
system.util.getProjectName()
```

Parameters

none

Returns

String - The name of the currently running project.

Scope

Client

Examples

```
# This code would display the name of the currently running project
system.gui.messageBox("You are running project: %s" % system.util.getProjectName())
```

9.13.11 system.util.getProperty

Description

Retrieves the value of a named system property. Some of the available properties are:
file.separator. The system file separator character. (for example, "/" (unix) or "\" (windows))
line.separator. The system line separator string. (for example, "\r\n" (carriage return, newline))
os.arch. Operating system architecture. (for example, "x86")
os.name. Operating system name. (for example, "Windows XP")
os.version. Operating system version. (for example, "5.1")
user.home. User's home directory.
user.name. User's account name.

Syntax

```
system.util.getProperty(propertyName)
```

Parameters

String `propertyName` - The name of the system property to get.

Returns

String - The value for the named property.

Scope

All

Examples

This script would store the contents of the Text Area component in the users home directory.

```
homeDir = system.util.getProperty("user.home")
sep = system.util.getProperty("file.separator")
path = "%s%smyfile.txt" %(homeDir, sep)
system.file.writeFile(path, event.source.parent.getComponent("Text Area").text)
```

© 2011 Inductive Automation

9.13.12 system.util.getReadTimeout

Description

Returns the read timeout in milliseconds for all client-to-gateway communication. This is the maximum amount of time allowed for a communication operation to complete. The default is 60,000ms (1 minute).

Syntax

```
system.util.getReadTimeout()
```

Parameters

none

Returns

int - The current read timeout, in milliseconds. Default is 60,000 (one minute)

Scope

Client

9.13.13 system.util.getSessionInfo

Description

Returns a PyDataSet holding information about all of the sessions (logged-in users) on the Gateway. Optional regular-expression based filters can be provided to filter the username or the username and the project returned.

The PyDataSet returned has these columns:

`username` (String)
`project` (String)
`address` (String)
`isDesigner` (Boolean)
`clientId` (String)
`creationTime` (Date)

Note that this function will not return all sessions across a cluster - only the cluster node that is being communicated with by the client who makes the call.

Syntax

```
system.util.getSessionInfo([usernameFilter] [, projectFilter])
```

Parameters

String `usernameFilter` - A filter string to restrict the list by username. * matches anything, ? matches one character. [optional]

String `projectFilter` - A filter string to restrict the list by project. * matches anything, ? matches one character. [optional]

Returns

PyDataSet - A dataset representing the Gateway's current sessions.

Scope

Client

Examples

Example 1:

© 2011 Inductive Automation

```
# This code would get the entire table of sessions and put it in an adjacent table
table = event.source.parent.getComponent("Table")
sessions = system.util.getSessionInfo()
table.data = system.db.toDataSet(sessions)
```

Example 2:

```
# This code would count the number of times a user named "billy" is logged in
sessions = system.util.getSessionInfo("billy")
system.gui.messageBox("Billy has %d sessions" % len(sessions))
```

9.13.14 system.util.getSystemFlags

Description

Returns an integer that represents a *bit field* containing information about the currently running system. Each bit corresponds to a public bitmask as defined below. See the examples for tips on how to extract the information in this bit field are in the examples. Note that the tag `[System]Client/System/SystemFlags` contains the same value.

- `system.util.DESIGNER_FLAG`. Set if running in the Designer. (1)
- `system.util.PREVIEW_FLAG`. Set if running in the Designer, and the Designer is in preview mode. (2)
- `system.util.CLIENT_FLAG`. Set if running as a Client. (4)
- `system.util.WEBSTART_FLAG`. Set if running as a Client in Web Start mode. (8)
- `system.util.APPLET_FLAG`. Set if running as a Client in Applet mode. (16)
- `system.util.FULLSCREEN_FLAG`. Set if running as a Client in full-screen mode. (32)
- `system.util.SSL_FLAG`. Set if communication to the Gateway is encrypted with SSL. (64)
- `system.util.MOBILE_FLAG`. Set if currently running a mobile-launched client. (128)

Syntax

```
system.util.getSystemFlags()
```

Parameters

none

Returns

int - The system flags integer.

Scope

Client

9.13.15 system.util.invokeAsynchronous

Description

This is an advanced scripting function. Invokes (calls) the given Python function on a different thread. This means that calls to `invokeAsynchronous` will return immediately, and then the given function will start executing asynchronously on a different thread. This is useful for long-running data intensive functions, where running them synchronously (in the GUI thread) would make the GUI non-responsive for an unacceptable amount of time.

WARNING: Under no circumstances should you ever do anything in the function that is invoked asynchronously that interacts with the GUI. This means things like window navigation, setting and

© 2011 Inductive Automation

getting component properties, showing error/message popups, etc. If you need to do something with the GUI in this function, this must be achieved through a call to `system.util.invokeLater`.

Syntax

system.util.**invokeAsynchronous(**function**)**

Parameters

PyObject `function` - A python function object that will get invoked with no arguments in a separate thread.

Returns

nothing

Scope

All

Examples

```
# This code would do some data-intensive processing, and then call
# back to the GUI to let it know that it is finished.
# We use default function parameters to pass the root container into these
# functions. (See a Python reference if you don't understand this)

def longProcess(rootContainer = event.source.parent):
   import system
   # Do something here with the database that takes a long time
   results = ...( something )
   # Now we'll send our results back to the UI
   def sendBack(results = results, rootContainer = rootContainer):
      rootContainer.resultsProperty = results
   system.util.invokeLater(sendBack)

system.util.invokeAsynchronous(longProcess)
```

9.13.16 system.util.invokeLater

Description

This is an advanced scripting function. Invokes (calls) the given Python function object after all of the currently processing and pending events are done being processed, or after a specified delay. The function will be executed on the GUI, or event dispatch, thread. This is useful for events like *propertyChange* events, where the script is called before any bindings are evaluated.

If you specify an optional time argument (number of milliseconds), the function will be invoked after all currently processing and pending events are processed *plus* the duration of that time.

Syntax

system.util.**invokeLater(**function **[**, delay**])**

Parameters

PyObject `function` - A Python function object that will be invoked later, on the GUI, or event-dispatch, thread with no arguments.

int `delay` - A delay, in milliseconds, to wait before the function is invoked. The default is 0, which means it will be invoked after all currently pending events are processed. [optional]

Returns

© 2011 Inductive Automation

nothing

Scope
Client

Examples

```
# The code in the update/refresh button uses the 'date' property on the two calendar components
# which are bound to the current_timestamp property on their parent. We want to simulate a butt
# press when the window opens, but only after the date properties' bindings have been evaluated

if event.propertyName == 'current_timestamp':
   # Define a function to click the button
   def clickButton(button = event.source.parent.getComponent('Refresh')):
      import system
      button.doClick()
      system.gui.messageBox("Button has been clicked!")

   # Tell the system to invoke the function after
   # the current event has been processed
   system.util.invokeLater(clickButton)
```

9.13.17 system.util.playSoundClip

Description

Plays a sound clip from a `wav` file to the system's default audio device. The `wav` file can be specified as a filepath, a URL, or directly as a raw byte[].

Syntax

system.util.**playSoundClip(**wavFile**)**

Parameters
String `wavFile` - A filepath or URL that represents a wav file

Returns
nothing

Scope
All

system.util.**playSoundClip(**wavBytes **[**, volume**] [**, wait**])**

Parameters
byte[] `wavBytes`
double `volume` - The clip's volume, represented as a floating point number between 0.0 and 1.0 [optional]
boolean `wait` - A boolean flag indicating whether or not the call to playSoundClip should wait for the clip to finish before it returns [optional]

Returns
nothing

Scope
All

system.util.**playSoundClip(**wavFile **[**, volume**] [**, wait**])**

Parameters

© 2011 Inductive Automation

String `wavFile` - A filepath or URL that represents a wav file
double `volume` - The clip's volume, represented as a floating point number between 0.0 and 1.0 [optional]
boolean `wait` - A boolean flag indicating whether or not the call to playSoundClip should wait for the clip to finish before it returns [optional]

Returns
nothing

Scope
All

Examples

Example 1:

```
# This code would play a sound clip at full volume that was located on the current host's files
# It will not return until the clip in finished playing
system.util.playSoundClip("C:\\sounds\\siren.wav")
```

Example 2:

```
# This code would pull a sound clip out of a BLOB field from a database, playing it asynchronou

soundData = system.db.runScalarQuery("SELECT wavBlob FROM sounds WHERE type='alert_high'")

system.util.playSoundClip(soundData, 0.5, 0)
```

9.13.18 system.util.queryAuditLog

Description

Queries an audit profile for audit history. Returns the results as a dataset.

This function accepts *keyword-style* invocation. See also: Functions / Keyword Invocation

Syntax

```
system.util.queryAuditLog(auditProfileName, startDate, endDate, actorFilter, actionFilter,
targetFilter, valueFilter, systemFilter, contextFilter)
```

Parameters

String `auditProfileName` - The name of the audit profile to pull the history from.
Date `startDate` - The earliest audit event to return. If omitted, the current time - 8 hours will be used.
Date `endDate` - The latest audit evnet to return. If omitted, the current time will be used.
String `actorFilter` - A filter string used to restrict the results by actor.
String `actionFilter` - A filter string used to restrict the results by action.
String `targetFilter` - A filter string used to restrict the results by target.
String `valueFilter` - A filter string used to restrict the results by value.
String `systemFilter` - A filter string used to restrict the results by system.
Integer `contextFilter` - A bitmask used to restrict the results by context. 0x01 = Gateway, 0x02 = Designer, 0x04 = Client.

Returns

Dataset - A dataset with the audit events from the specified profile that match the filter arguments.

© 2011 Inductive Automation

Scope
Client

9.13.19 system.util.retarget

Description

This function allows you to programmatically 'retarget' the Client to a different project and/or different Gateway. You can have it switch to another project on the same Gateway, or another gateway entirely, even across a WAN. This feature makes the vision of a seamless, enterprise-wide SCADA application a reality.

The retarget feature will attempt to transfer the current user credentials over to the new project / Gateway. If the credentials fail on that project, the user will be prompted for a valid username and password, with an option to cancel the retargeting and return to the original project. One valid authentication has been achieved, the currently running project is shut down, and the new project is loaded.

You can pass any information to the other project through the `parameters` dictionary. All entries in this dictionary will be set in the global scripting namespace in the other project. Even if you don't specify any parameters, the system will set the variable `_RETARGET_FROM_PROJECT` to the name of the current project and `_RETARGET_FROM_GATEWAY` to the address of the current Gateway.

Syntax

system.util.**retarget(**projectName **[**, gatewayAddress**] [**, params**] [**, startupWindows**])**

Parameters

String `projectName` - The name of the project to retarget to.
String `gatewayAddress` - The address of the Gateway that the project resides on. If omitted, the current Gateway will be used. Format is: "host:httpPort:sslPort/contextName" [optional]
PyDictionary `params` - A dictionary of parameters that will be passed to the new project. They will be set as global variables in the new project's Python scripting environment. [optional]
String[] `startupWindows` - A list of window names to use as the startup windows. If omitted, the project's normal startup windows will be opened. If specified, the project's normal startup windows will be ignored, and this list will be used instead. [optional]

Returns
nothing

Scope
Client

Examples

Example 1:

```
# This code would switch to a project named 'TankControl' on the same Gateway
# as the currently running project
system.util.retarget("TankControl")
```

Example 2:

```
# This code would switch to a project named 'TankControl' on a
# Gateway located at a different IP address running on port 8080, and
# would open the window named "Graph", and set a global jython variable in the new
```

© 2011 Inductive Automation

```
# project named "retargetOccured" to the value 1 (one).
system.util.retarget("TankControl", "10.30.2.33:8088/main", {"retargetOccured":1}, ["Graph"])
```

Example 3:

```
# This code would be put in a button in the target that was retargetted to,
# and act as a 'back' button, that would retarget back to the original project.

global _RETARGET_FROM_PROJECT
global _RETARGET_FROM_GATEWAY

# _RETARGET_FROM_GATEWAY is formatted like 'http://10.1.10.1:8088/main', so you have to remove
system.util.retarget(_RETARGET_FROM_PROJECT, _RETARGET_FROM_GATEWAY[7:])
```

9.13.20 system.util.setConnectTimeout

Description

Sets the connect timeout for client-to-gateway communication. Specified in milliseconds.

Syntax

```
system.util.setConnectTimeout(connectTimeout)
```

Parameters

int `connectTimeout` - The new connect timeout, specified in milliseconds.

Returns

nothing

Scope

Client

Examples

```
# This code would set the current connect timeout to 30 seconds
system.util.setConnectTimeout(30000)
```

9.13.21 system.util.setConnectionMode

Description

Sets the connection mode for the client session. Normally a client runs in mode 3, which is read-write. You may wish to change this to mode 2, which is read-only, which will only allow reading and subscribing to tags, and running SELECT queries. Tag writes and INSERT / UPDATE / DELETE queries will not function. You can also set the connection mode to mode 1, which is disconnected, all tag and query features will not work.

Syntax

```
system.util.setConnectionMode(mode)
```

Parameters

int `mode` - The new connection mode. 1 = Disconnected, 2 = Read-only, 3 = Read/Write.

Returns

nothing

Scope

© 2011 Inductive Automation

Client

Examples

This example, which could go in a project's startup script, would check the current username and set the connection mode to read-only if it is the "guest" user.

```
username = system.security.getUsername()
if "guest" == username.lower():
   # Set "guest" user to read-only mode
   system.util.setConnectionMode(2)
else:
   system.util.setConnectionMode(3)
```

9.13.22 system.util.setReadTimeout

Description

Sets the read timeout for client-to-gateway communication. Specified in milliseconds.

Syntax

```
system.util.setReadTimeout(readTimeout)
```

Parameters

int `readTimeout` - The new read timeout, specified in milliseconds.

Returns

nothing

Scope

Client

© 2011 Inductive Automation

Index

- 2 -

2-State Button 234

- A -

Aggregation Mode 161
Anchored Layout 155
Animation, using Timers 411
app.* 124
Applet Size 123
Arrow Component 397
Audio Playback 409
Auto-Login 123
Auto-Refresh 109

- B -

Bar Chart 349
Barcode component 287
Base Rate 124
Bidirectional Bindings 159
Blue Properties 150
Bold Properties 150
Box and Whisker Chart 362
Button Component 231

- C -

Caching Windows 142
Calculated Pens 335
Calendar Component 367
Centered Components 155
Chart Component 345
Checkbox Component 253
Circle Component 388
Classic Chart Component 345
Client Memory 123
Client Menubar Appearance 124
Client Poll Rate 120
Collapsible Palette 147
Column Selector Component 418
Comm Off 106
Comm Read/Write 106
Comm Read-Only 106
Comments Panel Component 330
Compass Component 294
Components
  Copying 148
  Creating 147
  Customizers 151
  Dynamic Properties 152
  Layout 155
  Moving 148
  Overlays 153
  Properties 150
  Resizing 148
  Security 174
  Styles 152
Container Component 404
Control Chart 335
CSV Export of Table 306, 317
Custom Palettes 147
Custom Properties 152
Customizers 151
Cylindrical Tank Component 279

- D -

Dashed Line 397
Data Types
  Color 150
  Dataset 150
  Date 150
  Double 150
  Float 150
  int 150
  Integer 150
  Long 150
  String 150
Database Pens 335
Databinding 157
Dataset
  Definition 150
  Scripting 187
Datatypes 150
Date Picker Component 370
Date Range Component 373
Date Spinner 212
Debugging scripts 107

© 2011 Inductive Automation

Default Color Mapping 122
Default Component Layout 122
Default Database Connection 120
Default Launch Mode 123
Default SQLTags Provider 120
Designer Shortcuts 149
Diagnostics 107
Digital Display Component 271
Dockable Panels 106
Document Viewer Component 301
Draw a Line 397
Dropdown Component 223
Dynamic Properties 152

- E -

Easy Chart 335
Editable Table 306, 317
Event Handlers
  Action Qualifiers 172
  Navigation 172
  Overview 164
  Set Property 172
  Set Tag Value 172
  SQL Update 172
event Object 165
Event Types
  actionPerformed 166
  cellEdited 166
  focusGained 166
  focusLost 166
  internalFrameActivated 166
  internalFrameClosed 166
  internalFrameClosing 166
  internalFrameDeactivated 166
  internalFrameOpened 166
  itemStateChanged 166
  keyPressed 166
  keyReleased 166
  keyTyped 166
  mouseClicked 166
  mouseDragged 166
  mouseEntered 166
  mouseExited 166
  mouseMoved 166
  mousePressed 166
  mouseReleased 166
  propertyChange 166
  repaint 166
event.source 165
Expert Properties 150
Expression Binding 162

- F -

Failure Handshake 128
Fallback Delay 159
Fallback Value 163
File Chooser 420
Formatted Text Field 215

- G -

Gantt Chart 364
Gateway Comm Mode 106
Gauge Component 290
getComponent 165
Go Back 172
Go Forward 172
Grouped Container 404
GW_COMM_OFF 106

- H -

Handshakes 128
Hiding a Project 123
Hiding the Exit Button 123
Hiding the Menubar 124
HOA Control 239
HTML Export of Table 306, 317
HTML Viewer Component 301

- I -

Image Component 273
Image Manager 108
Images 108
Indirect Bindings 159
Initial Gateway Comm Mode 121
IPCamera Component 303

- J -

Java Web Start Homepage 123

© 2011 Inductive Automation

Java Web Start Vendor 123
Java2D 407

- K -

Keyboard Shortcuts 149

- L -

Label Component 261
Latched Button 242
Launch Icon 123
Layout 155
LED Display Component 271
Level Indicator Component 281
Line Component 397
Line-Wrap 221
List Component 314
Log Viewer 107
Login Screen Settings 123

- M -

Meter Component 290
Minimum Size 124
MJPEG Video 303
Modules 41
Momentary Button 246
Multi-Line Text Editor 221
Multi-State Button 239
Multi-State Indicator 268

- N -

Navigation 144
Netcam Component 303
Nudge Distances 122
Number Spinner 212
Numeric Label Component 264
Numeric Text Editor 209

- O -

One-Shot (Latched) Button 242
Output Console 107
Oval Component 388
Overlays 153

- P -

Paintable Canvas 407
Palettes 147
Passing Parameters (Windows) 145
Password Field Component 218
PDF File Viewer 422
PDF Report Component 413
Pens 335
Performance 107
Perspectives 106
Pie Chart 358
Pipe Component 400
Playing Audio 409
Polling Base Rate 124
Polling Options 159
Polygon Component 394
Popup Calendar Component 370
Preview Mode 138
print keyword (Python) 107
Progress Bar 276
Projects
  Auditing 120
  Authentication 120
  Creating 54, 105
  Deleting 54
  Opening 105
  Securing 173
Property Binding 157
Property Binding Types
  DB Browse 162
  Expression 162
  Indirect Tag 161
  Property 162
  SQL Query 163
  SQLTags Historian 161
  Tag 160
Publish Mode 122
Pushbutton Component 231

- Q -

Quality Overlays 153
Query Base Rate 124
Query Browser 109

© 2011 Inductive Automation

- R -

Radio Button Component 255
Rectangle Component 391
Relative Layout 155
Relative Rate 159
Required Roles 120
Reset panels 106
Roles 173
Row Selector Component 415
RTF Viewer Component 301

- S -

Script Modules 124
Signal Generator 412
Slider Component 228
Sound Playback 409
SPC Chart 335
Spinner Component 212
SQLTags 32
SQLTags Historian 32
SQLTags Historian Pens 335
SQLTags Security 173
Square Component 391
Stale Overlay 153
Standard Properties 150
Star Component 394
Status Chart 354
Stored Procedures
  Stored Procedure Group 138
Styles Customizer 152
Success Handshake 128

- T -

Tabbed Palette 147
Table Component 306, 317
Tabstrip Component 258
Tank Component 279
Text Area Component 221
Text Field Component 206
Thermometer Component 298
Thread Viewer 107
Timer Component 411
Timezone Behavior 122
Toggle Button 234
Touch Screen Mode 122
Touch Screen Support 154
Touchscreen Support 154
Transaction Groups
  Block 136
  Historical 137
  Standard 135
  Stored Procedure Group 138
Treeview Component 326
Trial Timeout Overlay 153
Triangle Component 394
Triggers 128

- V -

Video Camera Component 303

- W -

WAV file 409
Window Committing 122
Window Workspace 138
Windows
  About Window 142
  Border Display Policy 142
  Caching 142
  Dock Position 142
  Docking 142
  Exporting 139
  Importing 139
  Layer 142
  Multiple Instances 144
  Notes 139
  Open on Startup 142
  Opening 144
  Organizing 139
  Passing Parameters 145
  Security 144
  Swapping 144
  Titlebar Display Policy 142
Workspace 105, 106

© 2011 Inductive Automation

Endnotes 2... (after index)

© 2011 Inductive Automation

Back Cover